

# CONDENSE: A Reconfigurable Knowledge Acquisition Architecture for Future 5G IoT

Dejan Vukobratovic, Dusan Jakovetic, Vitaly Skachek, Dragana Bajovic, Dino Sejdinovic, Gunes Karabulut Kurt, Camilla Hollanti, and Ingo Fischer

**Abstract**—In forthcoming years, the Internet of Things (IoT) will connect billions of smart devices generating and uploading a deluge of data to the cloud. If successfully extracted, the knowledge buried in the data can significantly improve the quality of life and foster economic growth. However, a critical bottleneck for realising the efficient IoT is the pressure it puts on the existing communication infrastructures, requiring transfer of enormous data volumes. Aiming at addressing this problem, we propose a novel architecture dubbed Condense (reconfigurable knowledge acquisition systems), which integrates the IoT-communication infrastructure into data analysis. This is achieved via the generic concept of network function computation: Instead of merely transferring data from the IoT sources to the cloud, the communication infrastructure should actively participate in the data analysis by carefully designed en-route processing. We define the Condense architecture, its basic layers, and the interactions among its constituent modules. Further, from the implementation side, we describe how Condense can be integrated into the 3rd Generation Partnership Project (3GPP) Machine Type Communications (MTC) architecture, as well as the prospects of making it a practically viable technology in a short time frame, relying on Network Function Virtualization (NFV) and Software Defined Networking (SDN). Finally, from the theoretical side, we survey the relevant literature on computing “atomic” functions in both analog and digital domains, as well as on function decomposition over networks, highlighting challenges, insights, and future directions for exploiting these techniques within practical 3GPP MTC architecture.

**Index Terms**—Internet of Things (IoT), Big Data, Network Coding, Network Function Computation, Machine learning, Wireless communications.

D. Vukobratovic is with the Department of Power, Electronics and Communications Engineering, University of Novi Sad, Serbia, e-mail: dejanv@uns.ac.rs.

D. Jakovetic is with the BioSense Institute, Novi Sad, Serbia, and with the Department of Mathematics and Informatics, Faculty of Sciences, University of Novi Sad, Serbia, email: djakovet@uns.ac.rs.

V. Skachek is with the Institute of Computer Science, University of Tartu, Estonia, email: vitaly.skachek@ut.ee.

D. Bajovic is with the BioSense Institute, Novi Sad, Serbia, and with the Department of Power, Electronics and Communication Engineering, University of Novi Sad, Serbia, email: dbajovic@uns.ac.rs.

D. Sejdinovic is with the Department of Statistics, University of Oxford, UK, email: dino.sejdinovic@stats.ox.ac.uk.

G. Karabulut Kurt is with the Department of Electronics and Communication Engineering, Istanbul Technical University, Turkey, email: gkurt@itu.edu.tr.

C. Hollanti is with the Department of Mathematics and Systems Analysis, Aalto University, Finland, email: camilla.hollanti@aalto.fi.

I. Fischer is with the Institute for Cross-Disciplinary Physics and Complex Systems (UIB-CSIC), Spain, email: ingo@ifisc.uib-csic.es.

D. Vukobratovic is financially supported by Rep. of Serbia TR III 44003 grant. V. Skachek is supported in part by the grant PUT405 from the Estonian Research Council. G. Karabulut Kurt is supported by TUBITAK Grant 113E294. C. Hollanti is financially supported by the Academy of Finland grants #276031, #282938 and #283262.

## I. INTRODUCTION

A deluge of data is being generated by an ever-increasing number of devices that indiscriminately collect, process and upload data to the cloud. An estimated 20 to 40 billion devices will be connected to the Internet by 2020 as part of the Internet of Things (IoT) [1]. IoT has the ambition to interconnect smart devices across cities, vehicles, appliances, connecting industries, retail and healthcare domains, thus becoming a dominant fuel for the emerging Big Data revolution [2]. IoT is considered as one of the key technologies to globally improve the quality of life, economic growth, and employment, with the European Union market value expected to exceed one trillion Euros in 2020 [3]. However, a critical bottleneck for the IoT vision is the pressure it puts on the existing communication infrastructures, by requiring transfer of enormous amounts of data. By 2020 IoT data will exceed 4.4 ZB (zettabytes) amounting to 10% of the global “digital universe” (compared to 2% in 2013) [4]. Therefore, a sustainable solution for IoT and cloud integration is one of the main challenges for contemporary communications technologies.

The state-of-the-art in IoT/cloud integration assumes uploading and storing all the raw data generated by IoT devices to the cloud. The IoT data is subsequently processed by cloud-based data analysis that aims to extract useful knowledge [5]. For majority of applications, this approach is inefficient since there is typically a large amount of redundancy in the collected data. As a preprocessing step prior to data analysis, projections to a much lower-dimensional space are often employed, essentially discarding large portions of data. With the growth of IoT traffic, the approach where communications and data analysis are separated will become unsustainable, necessitating a fundamental redesign of IoT communications.

In this work, we propose a generic and reconfigurable IoT architecture capable of adapting the IoT data transfer to the subsequent data analysis. We refer to the proposed architecture as Condense (reconfigurable knowledge acquisition systems). Instead of merely transferring data, the proposed architecture provides an active and reconfigurable service leveraged by the data analysis process. We identify a common generic interface between data communication and data analysis: *the function computation*, and we distinguish it as a core Condense technology. Instead of communicating a stream of data units from the IoT devices to the cloud, the proposed IoT architecture processes the data units *en-route* through a carefully designed process to deliver a stream of network function evaluations stored in the cloud. In other words, the Condense architecture

does not transfer all the raw data across the communications infrastructure, but only what is needed from the perspective of the current application at hand.

To illustrate the idea with a toy example, consider a number of sensors which constitute a fire alarm system, e.g., [6], [7]. Therein, we might only be interested in the maximal temperature across the sensed field, and not in the full sensors readings vector. Therefore, it suffices to deliver to the relevant cloud application only an evaluation of the maximum function applied over the sensors readings vector; Condense realizes this maximum function as a composition of “atomic” functions implemented across the communications infrastructure.

We describe how to implement the proposed approach explained above in the concrete third generation partnership project (3GPP) Machine Type Communications (MTC) architecture [8]. The 3GPP MTC service is expected to contribute a dominant share of the IoT traffic via the upcoming fifth generation (5G) mobile cellular systems, thus providing an ideal setup for the demonstration of Condense concepts. We enhance the 3GPP MTC architecture with the network function computation (NFC) – a novel envisioned MTC-NFC service. We define the layered Condense architecture comprised of three layers: i) atomic function computation layer, ii) network function computation layer, and iii) application layer, and we map these layers onto the 3GPP MTC architecture. In the lowermost atomic function computation (AFC) layer, carefully selected atomic modules perform local function computations over the input data. The network function computation layer orchestrates the collection of AFC modules into the global network-wide NFC functionality, thus evaluating non-trivial functions of the input data as a coordinated composition of AFCs. Furthermore, the NFC layer provides a flexible and reconfigurable MTC-NFC service to the topmost application layer, where cloud-based data analysis applications directly exploit the outputs of the NFC layer. Throughout the system description, we provide a review of the theoretical foundations that justify the proposed architecture and point to the tools for the system design and analysis. Finally, we detail practical viability of incorporating NFC services within 3GPP MTC service, relying on emerging concepts of Network Function Virtualization (NFV) [9], [10] and Software Defined Networking (SDN) [11], [12]; this upgrade is, thanks to the current uptake of the SDN/NFV concepts, achievable within a short time frame.

This paper is somewhat complementary with respect to other works that consider architectures for 5G IoT communications. For example, reference [13] focuses on machine-type multicast services to ensure end-to-end reliability, low latency and low energy consumption of MTC traffic (including both up and downlinks). Reference [14] provides a detailed analysis of integration of 5G technologies for the future global IoT, both from technological and standardization aspects. However, while existing works consider making communication of the MTC-generated *raw data* efficient, here we aim to improve the overall system efficiency through communicating over the network infrastructure only the application-requested *functions over data*. In other words, this paper describes how we can potentially exploit decades of research on function computation

and function decomposition over networks within the concrete, practical and realizable knowledge acquisition system for the IoT-generated data. In particular, we review the main results on realizing (atomic) function computation in the analog (wireless and optical) and digital domains, as well as on function evaluation and decomposition over networks, including the work on sensor fusion, e.g., [6], [7], network coding for computing [15], [16]–[19], and neural networks [20]–[24]. While this paper does not provide novel contributions to these fields, it identifies and discusses main challenges in applying them within the practical 3GPP MTC architecture, and it points to interesting future research directions.

**Paper organization.** The rest of the paper is organized as follows. In Sec. II, we review the state-of-the-art 3GPP MTC architecture, briefly present SDN/NFV concepts, and give notational conventions. In Sec. III, we introduce the novel layered Condense architecture that, through the rest of the paper, we integrate into the 3GPP MTC architecture. In Sec. IV, we describe the atomic function computation layer that defines the basic building block of the architecture, distinguishing between the analog (or *in-channel*) AFC and digital (*in-node*) AFC modules. The theoretical fundamentals and practical aspects of the NFC layer are presented in Sec. V. In Sec. VI, the interaction between the application layer and the NFC layer is discussed, where several application layer examples are presented in detail. Further implementation issues are discussed in Sec. VII, and the paper is concluded in Sec. VIII.

## II. BACKGROUND AND PRELIMINARIES

Subsection II-A reviews the current 3GPP MTC architecture, Subsection II-B gives background on software defined networking (SDN) and network function virtualization (NFV), while Subsection II-C defines notation used throughout the rest of the paper.

### A. The 3GPP MTC Architecture

Machine Type Communications (MTC) is an European Telecommunications Standards Institute (ETSI)-defined architecture that enables participating devices to send data to each other or to a set of servers [8]. While ETSI is responsible for defining the generic MTC architecture, specific issues related with mobile cellular networks are addressed in 3GPP standardization [25]. 3GPP MTC is first included in Release 10 and will evolve beyond current 3GPP Long Term Evolution (LTE)/LTE-Advanced Releases into the 5G system [26].

Fig. 1 illustrates the 3GPP MTC architecture. It consists of: i) the MTC device domain containing MTC devices that access MTC service to send and/or receive data, ii) the network domain containing network elements that transfer the MTC device data, and iii) the MTC application domain containing MTC applications running on MTC servers. MTC devices access the network via Radio Access Network (RAN) elements: base stations (eNB: eNodeB) and small cells (HeNB: Home-eNodeB). Packet data flows follow the Evolved Packet Core (EPC) elements: HeNB Gateway (HeNB-GW), Service Gateway (S-GW) and Packet Gateway (P-GW), until they

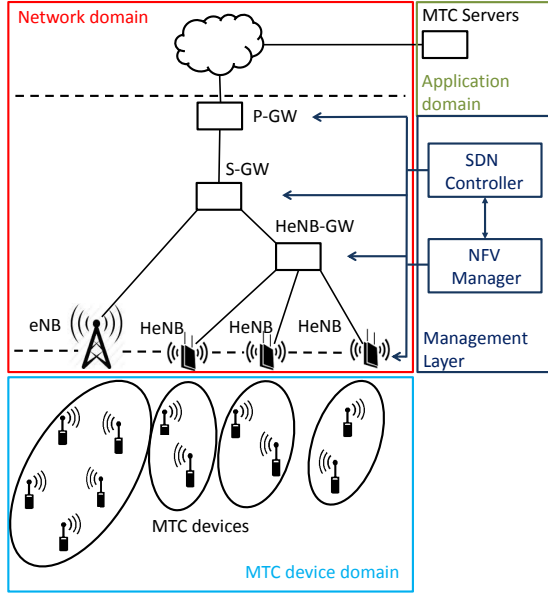


Fig. 1. The 3GPP MTC architecture.

reach either a mobile operator MTC server or a third party MTC server via the Internet. In this work, we address MTC device data processing and focus on the data plane while ignoring the control plane of the 3GPP architecture.

Abstracted to its essence, the current 3GPP MTC approach in the context of IoT/cloud integration is represented by three layers (Fig. 1). The MTC device domain, or data layer, contains billions of devices that generate data while interacting with the environment. The network domain, or communication layer, provides mere data transfer services to the data layer by essentially uploading the generated data to the cloud in its entirety. The application domain, or application layer, contains data centres running MTC servers which provide storage and processing capabilities. MTC applications running in data centres enable, e.g., machine learning algorithms to extract knowledge from the collected data. In this paper, we challenge this 3GPP MTC layered structure and propose a novel Condense layered architecture described in Sec. III.

### B. Software Defined Networking (SDN) and Network Function Virtualization (NFV)

SDN and NFV are novel concepts in networking research that increase network flexibility and enable fast implementation of new services and architectures. Both SDN and NFV are under a current consideration for future integration in the 3GPP cellular architecture [27], [28]. Although not yet part of the 3GPP architecture, in Fig. 1, we present main NFV/SDN management entities: the NFV manager and the SDN controller, as they will be useful for the description of the Condense architecture.

SDN is a novel network architecture that decouples the control plane from the data plane [11], [29]. This is achieved by centralizing the traffic flow control where a central entity, called the SDN controller, manages the physical data forwarding process in SDN-enabled network nodes. The SDN con-

troller remotely manages network nodes and flexibly controls traffic flows at various flow granularities. In other words, the SDN controller can easily (re)define forwarding rules for data flows passing through an SDN network node. Using SDN, various network services are able to quickly re-route their data flows, adapting the resulting (virtual) network topology to their needs.

NFV is another recent trend in networking where, instead of running various network functions (e.g., firewalls, NAT servers, load balancers, etc.) on dedicated network nodes, the network hardware is virtualized to support software-based implementations of network functions [10]. This makes network functions easy to instantiate anywhere across the network when needed. Multiple instances of network functions are jointly administered and orchestrated by the centralized NFV management.

NFV and SDN are complementary concepts that jointly provide flexible and efficient service chaining: a sequence of data processing tasks performed at different network nodes [30]. The NFV manager has the capability to actually instantiate the targeted (atomic) function computations at each node in the network. Similarly, SDN has the power to steer data flows and hence establish a desired (virtual) network topology which supports the desired network-wide computation. This feature will be fundamental for a fast implementation and deployment of the Condense architecture, as detailed in the rest of the paper. For more details about SDN/NFV concepts in 3GPP networks, we refer the interested reader to [27].

### C. Notational preliminaries

Throughout, we use bold symbols to denote vectors, where  $\ell$ -th entry of a vector  $\mathbf{x}$  of length  $L$  is denoted by  $x[\ell]$ ,  $\ell = 1, \dots, L$ . We denote by  $\mathbb{R}$  the set of real numbers, and by  $\mathbb{R}^Q$  the  $Q$ -dimensional real coordinate space. A finite field is denoted by  $\mathbb{F}$ , a finite alphabet (finite discrete set) by  $\mathbb{A}$ , and by  $\mathbb{A}^Q$  the set of  $Q$ -dimensional vectors with the entries from  $\mathbb{A}$ . Symbol  $|\cdot|$  denotes the cardinality of a set. We deal with vectors  $\mathbf{x} \in \mathbb{R}^Q$ ,  $\mathbf{x} \in \mathbb{A}^Q$ , and also  $\mathbf{x} \in \mathbb{F}^Q$ , and it is clear from context which of the three cases is in force. Also, addition and multiplication over both  $\mathbb{R}$  and  $\mathbb{F}$  are denoted in a standard way – respectively as  $+$  and  $\cdot$  (or the multiplication symbol is simply omitted), and again the context clarifies which operation is actually applied.

We frequently consider a directed acyclic graph  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ , where  $\mathcal{V}$  denotes the set of nodes, and  $\mathcal{E}$  the set of directed edges (arcs). An arc from node  $u$  to node  $v$  is denoted by  $u \rightarrow v$ . Set  $\mathcal{V} = \mathcal{S} \cup \mathcal{A} \cup \mathcal{D}$ , where  $\mathcal{S}$ ,  $\mathcal{A}$ , and  $\mathcal{D}$  denote, respectively, the set of source nodes, atomic nodes, and destination nodes. We let  $\mathcal{S} \cap \mathcal{D} = \emptyset$ . We also introduce  $N = |\mathcal{S}|$ ,  $M = |\mathcal{A}|$ , and  $R = |\mathcal{D}|$ . As we will see further ahead, source nodes correspond to MTC devices ( $N$  data generators), atomic nodes correspond to the 3GPP communication infrastructure nodes which implement atomic functions ( $M$  atomic nodes), and destination nodes are MTC servers in data centers which are to receive the desired function computation results ( $R$  destination nodes). We index an arbitrary node in  $\mathcal{S}$  by  $s$ , and similarly we write  $a \in \mathcal{A}$ , and  $d \in \mathcal{D}$ . When we do

not intend to make a distinction among  $\mathcal{S}$ ,  $\mathcal{A}$ , and  $\mathcal{D}$ , we index an arbitrary node by  $v \in \mathcal{V}$ . For each node  $v \in \mathcal{V}$ , we denote by  $\mathcal{V}_{\text{in}}^{(v)}$  its in-neighborhood, i.e., the set of nodes  $v'$  in  $\mathcal{V}$  such that the arc  $v' \rightarrow v$  exists. Analogously,  $\mathcal{V}_{\text{out}}^{(v)}$  denotes the node  $v$ 's out-neighborhood. As we frequently deal with in-neighborhoods, we will simply write  $\mathcal{V}^{(v)} \equiv \mathcal{V}_{\text{in}}^{(v)}$ . We call the in-degree of  $v$  the cardinality of  $\mathcal{V}_{\text{in}}^{(v)}$ , and we analogously define the out-degree. Although not required by the theory considered ahead, just for simplicity of notation and presentation, all sections except Section V consider the special case where  $\mathcal{G}$  is a directed rooted tree with  $N$  sources and a single destination. Pictorially, we visualize  $\mathcal{G}$  as having the source nodes  $\mathcal{S}$  at the bottom, and the destination node  $d$  at the top (see Sec. V, Fig. 6, right-hand side). In the case of a directed rooted tree graph  $\mathcal{G}$ , the leaf nodes' set of  $v$  coincides with its in-neighborhood  $\mathcal{V}^{(v)}$ , and all nodes except the destination nodes have the out-degree one, the destination node having the out-degree zero.

We index (vector) quantities associated with sources  $s \in \mathcal{S}$  through subscripts, i.e.,  $\mathbf{x}_s$  is the source  $s$ 's vector. When considering a generic directed acyclic graph  $\mathcal{G}$  (Section V), we associate to each arc  $u \rightarrow v$  a vector quantity  $\mathbf{x}^{(u \rightarrow v)}$ . With directed rooted trees (Sections III, IV, and VI), each node (except the destination node) has the out-degree one; hence, for simplicity, we then use node-wise (as opposed to edge-wise) notation, i.e., we index quantity  $\mathbf{x}^{(u \rightarrow v)}$  as  $\mathbf{x}^{(u)}$ . Note that this notation is sufficient as, with directed rooted trees, there is only a single arc outgoing a (non-destination) node. When needed, time instances are denoted by  $t = 1, 2, \dots, T$ ; a vector associated with source  $s$  and time  $t$  is denoted by  $\mathbf{x}_{s,t}$ ; similarly, we use  $\mathbf{x}_t^{(v)}$  for non-source nodes.

### III. CONDENSE ARCHITECTURE: IOT/CLOUD INTEGRATION FOR 5G

In this section, we present the Condense architecture that upgrades the 3GPP MTC architecture with the concept of network function computation (NFC). NFC creates a novel role 3GPP MTC service should offer: instead of communicating raw data, it should deliver function computations over the data, providing for a novel MTC-NFC service. The NFC design should be generic, flexible and reconfigurable to meet the needs of increasing number of MTC applications that extract knowledge from MTC data. For most applications, indiscriminate collection of MTC data is extremely wasteful and MTC-NFC service may dramatically reduce MTC traffic while preserving operational efficiency of MTC applications.

The Condense architecture challenges the conventional division into data, communications and application layer (Sec. 2A). Instead, we propose a novel architecture consisting of: i) atomic function computation (AFC) layer, ii) network function computation (NFC) layer, and iii) application layer. In this section, we provide a high-level modular description of the architecture by carefully defining its basic building blocks (modules). In the following three sections, we delve into details of each layer and provide both theoretical justifications and implementation discussion that motivated this work.

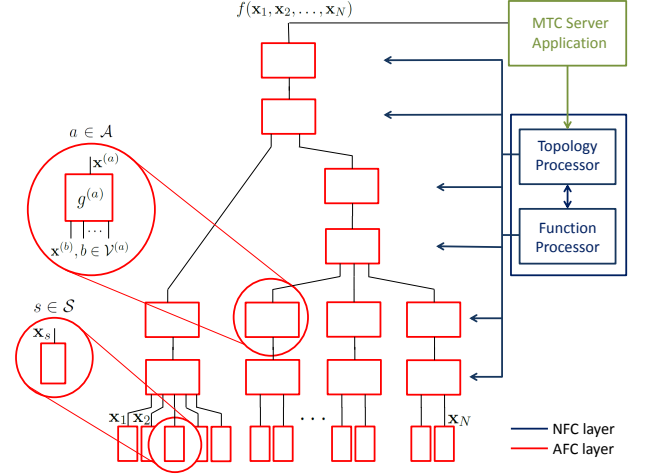


Fig. 2. Condense MTC-NFC architecture.

#### A. CONDENSE Architecture: Modules and Layers

The Condense architecture is presented in Fig. 2. It consists of an interconnected collection of basic building blocks called AFC modules. Each AFC module evaluates an (atomic) function over the input data packets and delivers an output data packet representing the atomic function evaluation. A generic AFC module may have multiple input and multiple output interfaces, each output interface representing a different AFC over the input data. The collection of interconnected and jointly orchestrated AFC modules delivers a network function computation over the source data packets. The resulting NFC evaluations are the input to application layer MTC server application.

Let us assume that an MTC network contains  $N$  MTC devices representing the set of source modules (or source nodes)  $\mathcal{S}$ . Source node  $s \in \mathcal{S}$  produces a message  $\mathbf{x}_s = (x_s[1], x_s[2], \dots, x_s[L])$  containing  $L$  symbols from a given alphabet  $\mathbb{A}$ . The message  $\mathbf{x}_s$  is transmitted at an output interface of the source module  $s$ . For simplicity, we assume that every source module has a single output interface.

In addition to the source nodes, the MTC network contains  $M$  AFC modules (or AFC nodes) representing the set  $\mathcal{A}$ . An arbitrary AFC node  $a \in \mathcal{A}$  has  $P$  input and  $Q$  output interfaces. For simplicity, unless otherwise stated, we will assume single-output AFC modules, i.e.,  $Q = 1$ . At input interfaces, the AFC node  $a$  receives the set of input data packets  $\{\mathbf{x}^{(b)}\}_{b \in \mathcal{V}^{(a)}}$ , while at the output interface, it delivers the output data packet  $\mathbf{x}^{(a)}$ . AFC node  $a$  associates an atomic function  $g^{(a)}$  to the output interface, where  $\mathbf{x}^{(a)} = g^{(a)}(\{\mathbf{x}^{(b)}\}_{b \in \mathcal{V}^{(a)}}$ . Finally, the MTC network contains  $R$  MTC servers (or destination nodes) representing the set of destination nodes  $\mathcal{D}$ .

The source nodes  $\mathcal{S}$ , AFC nodes  $\mathcal{A}$  and destination nodes  $\mathcal{D}$  are interconnected into an NFC graph  $\mathcal{G} = (\mathcal{V} = \mathcal{S} \cup \mathcal{A} \cup \mathcal{D}, \mathcal{E})$ , where  $\mathcal{V}$  is the set of nodes (modules) and  $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$  is the set of edges, i.e., connections between modules. For simplicity, unless otherwise stated, we restrict our attention to directed rooted trees (also called in-trees), where each edge is oriented

towards the root node<sup>1</sup>. Source nodes  $\mathcal{S}$  represent leaves of  $\mathcal{G}$ . The set of all edges in the graph is completely determined by the set of child nodes of all AFC and destination nodes. We let  $\mathcal{V}^{(v)}$  denote the set of child nodes of an arbitrary node  $v$ . The collection of sets  $\{\mathcal{V}^{(v)}\}_{v \in \mathcal{V}}$  fully describes the set of connections between modules.

Finally, we introduce the control elements: *topology processor* and *function processor*, that organize AFC modules into a global NFC evaluator. Based on the MTC server application requirements, the topology and function processors reconfigure the AFC modules to provide a requested MTC-NFC service. In particular, the function processor decomposes a required global NFC into a composition of local AFCs and configures each AFC module accordingly. In other words, based on the requested global network function  $f(\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N)$ , the function processor defines a set of atomic functions  $\{g^{(a)}\}_{a \in \mathcal{A}}$  and configures the respective AFC modules. Similarly, by defining the graph  $\mathcal{G}$  via the set  $\{\mathcal{V}^{(v)}\}_{v \in \mathcal{V}}$  and by configuring each AFC node accordingly, the topology processor will interconnect AFC modules into a directed graph of MTC data flows. The topology and function processor are key NFC layer entities. They manage, connect and orchestrate the AFC layer entities, i.e., source modules and AFC modules.

#### B. CONDENSE Architecture: Implementation

The above described abstract Condense architecture can be mapped onto the 3GPP MTC architecture. We present initial insights here, while details are left for the following sections.

The AFC layer is composed of AFC modules that evaluate atomic functions. Examples of atomic functions suitable for AFC implementations are the addition, modulo addition, maximum/minimum, norm, histogram, linear combination, threshold functions, etc. Atomic functions can be evaluated straightforwardly in the digital domain using digital processing in network nodes. In addition to that, atomic functions could be realized by exploiting superposition of signals in the analog domain. Thus, we consider two types of AFC modules: i) Analog-domain AFC (A-AFC), and ii) Digital-domain AFC (D-AFC) modules.

An A-AFC, also referred to as an in-channel AFC, harnesses interference in a wireless channel or signal combining in an optical channel to perform atomic function evaluations. An example of the technology that can be easily integrated as an A-AFC module is the Physical Layer Network Coding (PLNC) [31], [32], where the corresponding atomic function is finite field addition, e.g., bit-wise modulo 2 sum in the case of the binary field.

A D-AFC, also referred to as in-node AFC, evaluates atomic functions in the digital domain using, e.g., reconfigurable hardware-based modules in the context of SDN-based implementation [29]. Alternatively, they can also be implemented using software-based virtual network functions in the context of a NFV-based implementation [10]. An example of the technology that can be easily integrated as a D-AFC module is the packet-level Random Linear Network Coding (RLNC)

[33], [34]. RLNC is a mature technology in terms of optimized software implementations (see, e.g., [35]) and it evaluates linear combinations over finite fields as atomic functions. We note that it has been recently proposed and demonstrated within the SDN/NFV framework [36], [37].

The NFC layer can be naturally implemented within the SDN/NFV architecture. In particular, the topology processor naturally fits as an SDN application running on top of the SDN controller within the SDN architecture. In addition, the function processor role may be set within an NFV manager entity, e.g., taking the role of the NFV orchestrator. Using the SDN/NFV framework, MTC-NFC service can be quickly set and flexibly reconfigured according to requests arriving from a diverse set of MTC applications.

#### IV. ATOMIC FUNCTION COMPUTATION LAYER

In this Section, we discuss theoretical and implementation aspects of realizing atomic functions within AFC modules. Subsection IV-A discusses the AFC modules operating in the analog domain, while Subsection IV-B considers digital domain AFCs.

##### A. Analog-domain Atomic Function Computation (A-AFC)

**Wireless-domain A-AFC: Theory.** An A-AFC module's functionality of computing functions over the incoming packets is based on harnessing interference, i.e., the superposition property of wireless channels. We survey the relevant literature on such function computation over wireless channels, finalizing the subsection with presenting current theoretical and technological capabilities.

The idea of harnessing interference for computation is investigated in terms of a joint source-channel communication scheme in [38], targeting to exploit multiple access channel characteristics to obtain optimal estimation of a target parameter from noisy sensor readings. Extensions of the analog joint source-channel communication are further investigated in the literature, see e.g., [39]–[42]. Following the impact of network coding ideas across the networking research, reference [31] proposes the concept of PLNC to increase throughput of wireless channels; PLNC essentially performs specific A-AFC computations (finite field arithmetics) in a simple two-way relay channel scenario. Computation of linear functions or, more precisely, random linear combinations of the transmitted messages over multiple access channels (MAC) has been considered in [43] and extended in [44]; therein, the authors propose the compute-and-forward (CF) transmission scheme for computing linear functions at the relays, who attempt to decode the received random message combinations (the randomness is induced by the fading channel coefficients) to integer combinations, which hence become lattice points in the original code lattice. After this, the relays forward the lattice points to the destination, who can then solve for the original messages provided that the received system of equations is invertible.

Finally, reference [45] addresses non-linear function computation over wireless channels (see also [43]). While it is intuitive that a linear combination of packets (signals) from

<sup>1</sup>We note that this restriction is for simplicity of presentation only; extension to directed acyclic graphs is straightforward and will be required in Sec. V.



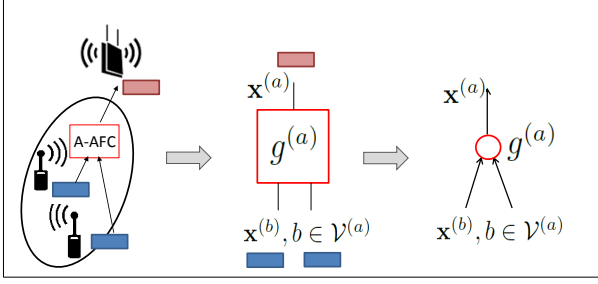


Fig. 3. Wireless-domain A-AFC module: representation via 3GPP elements (left), modules (middle) and NFC graph nodes (right).

multiple sources can be obtained through a direct exploitation of interference, more general, non-linear functions can also be computed through introducing a non-linear (pre-)processing of packets prior to entering the wireless medium, and their (post-)processing after the pre-processed signals have been superimposed in the wireless channel.

Following [45], we now describe in more detail how this non-linear function computation works – and hence how the A-AFC modules (in principle) operate. Assume that length- $L$  source node data packets  $\mathbf{x}_s = (x_s[1], x_s[2], \dots, x_s[L])$ ,  $s \in \mathcal{V}^{(a)} \subseteq \mathcal{S}$ , arrive at the input interfaces of an AFC node  $a$ . The packets are first pre-processed by the source node (MTC device) through a pre-processing function  $\varphi_s(\mathbf{x}_s)$ . The result is the transmitted symbol sequence  $\mathbf{y}_s = (y_s[1], y_s[2], \dots, y_s[L])$ , where  $y_s[\ell] = \varphi_s(x_s[\ell])$ ,  $\ell = 1, 2, \dots, L$ . Assuming a block-fading wireless channel model for narrowband signals, the received sequence can be modelled as  $\mathbf{r}^{(a)} = (r^{(a)}[1], r^{(a)}[2], \dots, r^{(a)}[L])$ , where:

$$r^{(a)}[\ell] = \sum_{s \in \mathcal{V}^{(a)}} h_s \cdot y_s[\ell], \quad \ell = 1, 2, \dots, L. \quad (1)$$

At the destination, a post-processing function  $\psi(\mathbf{r}^{(a)})$  is used to obtain  $\mathbf{x}^{(a)}$ , where  $x^{(a)}[\ell] = \psi(r^{(a)}[\ell])$ . Therefore, symbol-wise, the A-AFC module  $a$  realizes computation of the following (possibly non-linear) function:

$$g^{(a)}(\{x_s[\ell]\}_{s \in \mathcal{V}^{(a)}}) = \psi \left( \sum_{s \in \mathcal{V}^{(a)}} h_s \varphi_s(x_s[\ell]) \right). \quad (2)$$

The class of functions computable via A-AFC modules, i.e., which are of form (2), are called nomographic, and they include important functions such as the arithmetic mean and the Euclidean norm [45].

Fig. 3 illustrates an A-AFC: its position in the real-world system (left), its representation as an A-AFC module (central), and as part of the NFC graph (right).

**Wireless-domain A-AFC: Implementation.** The above framework implies that an A-AFC module is physically spread across all input devices and the output device connected to the A-AFC module, as illustrated in Fig. 4. At the input devices (e.g., MTC devices), an appropriate input A-AFC digital interface needs to be defined that accepts input data packets and implements pre-processing function  $\varphi(\cdot)$  before the signal is transmitted into the channel. Similarly, at the output device,

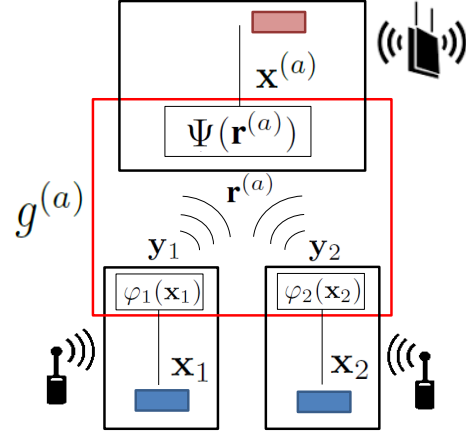


Fig. 4. Wireless-domain A-AFC module: Functional diagram.

e.g., small base station (HeNB), an appropriate output A-AFC digital interface needs to be defined that delivers output data packets after the signal received from the channel is post-processed using  $\psi(\cdot)$ . We also note that, although above we assume input nodes to A-AFC module are source nodes (MTC devices), wireless-domain A-AFC module can be part of the wireless backhaul network, e.g., connecting several HeNBs to the eNB.

**Challenges and Future Directions.** In the current state-of-the-art, A-AFC is investigated in the context of joint computation and communications in wireless sensor networks. Current research works are limited in terms of the computed functions, such as addition, multiplication, norm, arithmetic/geometric means, and are also limited in scope, as they are only targeting the wireless – and not optical – communication links. Design and implementation of generic A-AFC in wireless setting which is adaptive to the channel conditions remains an open problem. Note that any such design should take practical implementation aspects into account, including channel estimation errors, timing and frequency offsets and quantization issues. Furthermore, the link qualities between network nodes, including adaptive schemes that select the computation nodes according to the robustness of communications between links, need to be considered to improve the reliability of the computed function outputs.

**Optical-domain A-AFC: Discussion.** If we consider the PLNC example, it is clear that wireless domain A-AFC modules are close to become a commercially available technology (see, e.g., [32]). The question that naturally arises is whether A-AFC modules can be implemented in optical channels within optical access networks such as passive optical networks (PON). This would further increase the richness of AFC layer and bring novel AFC modules into the MTC-NFC network. Here, we briefly comment on the status of optical-domain function computation.

Recent works analyzed applicability of network coding of data packets within PONs in some simple scenarios [46] [47]. However, in contrast to the above vision of A-AFC modules, in these works signals are not “in-channel” combined, rather,

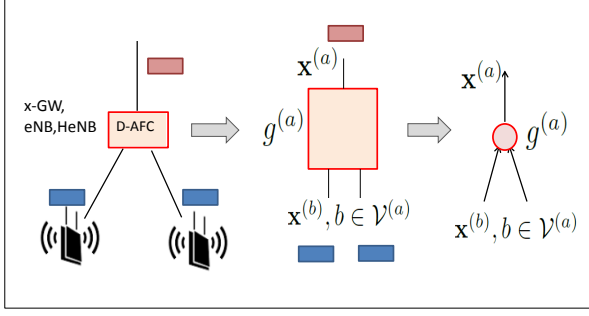


Fig. 5. D-AFC module: representation via 3GPP elements (left), modules (middle) and NFC graph nodes (right).

network coding is done at the end-nodes, in the digital domain.

Information processing in the photonic domain has been envisioned in the 1970s. But implementations of digital optical computing could not keep pace with the development of electronic computing. Nevertheless, with advances in technology, the role of optics in advanced computing has been receiving reawakened interest [48]. Moreover, unconventional computing techniques, in particular reservoir computing (RC), find more and more interest and are being implemented in different photonic hardware. RC is a neuro-inspired concept for designing, learning, and analysing recurrent neural networks neural networks where, unlike the most popular feed-forward neural networks, the interconnection network of neurons possesses cycles (feedback loops). A consequence of the presence of loops is, as pointed out in [49], that recurrent neural networks can process and account for temporal information at their input. A recent breakthrough was a drastic simplification of the information-processing concept of reservoir computing (RC) in terms of hardware requirements [50]. The appeal of RC therefore resides not only in its simple learning, but moreover in the fact that it enables simple hardware implementations. Complex networks can be replaced by a single or a few photonic hardware nodes with delayed feedback loops [51], [52], [53]. Different tasks, including spoken digit recognition, nonlinear time series prediction and channel equalization have been performed with excellent performance, speed and high energy efficiency [53], [54]. Beyond these first successes, meanwhile, using simple hardware, learning approaches including RC, extreme learning machines and back-propagation learning of recurrent neural networks have been demonstrated [55], illustrating the flexibility and potential of this approach.

### B. Digital-domain Atomic Function Computation (D-AFC)

D-AFC modules evaluate atomic functions in the digital domain, within the network nodes such as base stations (eNB or HeNB) and core network gateways (HeNB-GW, S-GW, P-GW). Although digital-domain in-node processing offers many possibilities for D-AFC implementation, here we address two possible options suitable for the SDN and NFV architectures.

The first option for D-AFC are reconfigurable hardware-based Field Programmable Gate Array (FPGA) platforms. FPGA platforms are frequently used in combination with high-speed networking equipment to perform various work-

intensive and high-throughput demanding functions over data packets such as packet filtering [56]. FPGAs are either integrated in network nodes as co-processing units, or can be easily attached as external units to network nodes via high-speed network interfaces. FPGAs offer flexible and reconfigurable high-throughput implementations of various linear or non-linear atomic functions. For example, implementing random linear combinations over input data packets in network nodes – as part of RLNC – is considered in several recent works [57], [58]. D-AFC implementations via FPGA platforms offer seamless integration in SDN concepts, because SDN data flows can be easily filtered and fed into either internal or external FPGA units. Depending on the application, FPGAs achieve speed increase over general processing units by factor of tens to hundreds. Note also that FPGAs can be reprogrammed and reconfigured in short time intervals (order of minutes).

The second possibility for efficient D-AFC is to use software-based implementations in high-level programming languages that run on general processing units, either in network nodes or externally on dedicated general-purpose servers [35]. This approach offers full flexibility for atomic function evaluation at the price of lower data processing throughput as compared with the FPGA approach. An example of a D-AFC implementation of random linear combinations over incoming data packets in the context of RLNC is given in [36], [37]. Software-based D-AFC implementations can be easily and remotely instantiated across the network nodes in a virtualized environment following NFV concepts.

Fig. 5 illustrates a D-AFC: its position in the real-world system (left), its representation as an D-AFC module (center), and as part of the NFC graph (right).

## V. NETWORK FUNCTION COMPUTATION LAYER

The NFC layer is responsible for configuring the Condense topology and assigning the appropriate atomic functions across the AFC modules, such that a desired network-wide function computation is realized. Subsection V-A discusses theoretical aspects (capabilities and limitations) of computing functions over networks, surveying the relevant literature on sensor fusion and network coding for computing. Subsection V-B describes a possible implementation of NFC functionalities within the 3GPP MTC system, through a more detailed view of SDN/NFV modules, i.e., the function and topology processors.

### A. Theoretical Aspects of NFC Layer

The need for mathematical theory of function computation in networks is advocated in [6], [7]. The authors discuss various challenges in sensor networks, and argue that computation of functions in a sensor network could lead to a lower data overhead, as well as to a reduced data traffic. For our toy example, in the fire alarm sensor network, we are only interested in the measurements of the highest temperature in the set of sensors. Alternatively, in monitoring temperature range in a green house, we might only be interested in the measurements of the average temperature from the set of sensors. Therefore, for various practical applications, it would be beneficial if the network node would be able to perform

basic (atomic) computation, which in the context of the whole network could lead to computation of more sophisticated functions in the destination nodes.

This subsection elaborates on the mathematical tools behind the realization of the Condense NFC layer. There is a number of works studying function computation over a network, which are available in the literature. The relevant work includes those in contexts of sensor fusion, network coding for computing, and neural networks. The two former work threads are discussed here, while the latter is discussed in Subsection VI-C. Hereafter, we mostly follow the framework defined in [15], [16], adapting notation to our needs here.

**Mathematical settings.** Consider a finite directed acyclic graph  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ , consisting of  $M$  AFC nodes belonging to set  $\mathcal{A}$ , a set of  $N$  sources (MTC devices)  $\mathcal{S}$ , and a set of  $R$  destinations  $\mathcal{D}$ , such that  $\mathcal{S} \cap \mathcal{D} = \emptyset$ .

The network uses a finite alphabet  $\mathbb{A}$ , called *network alphabet*. Each source  $s$  generates  $K$  random symbols  $\sigma_s[1], \sigma_s[2], \dots, \sigma_s[K] \in \mathbb{A}$ . Here, we say that the source symbol  $\sigma_s[k]$  belongs to the  $k$ -th generation of the source symbols.

We assume that each packet sent over a network link is a vector of length  $L$  over  $\mathbb{A}$ . Suppose that each of the  $R$  destination nodes requests computation of a (vector-valued) function  $f$  of the incoming MTC device vectors  $\sigma_s$ ,  $s = 1, \dots, N$ . The target vector function is of the form  $f : \mathbb{A}^{N \cdot K} \rightarrow \mathbb{B}^K$ , where  $\mathbb{B}$  is a function alphabet, and each component function  $f : \mathbb{A}^N \rightarrow \mathbb{B}$  is of the same form, applied to each source's  $k$ -th symbol,  $k = 1, \dots, K$ . More precisely, we wish to compute  $f(\sigma_1[k], \dots, \sigma_N[k])$ ,  $k = 1, \dots, K$ .

With each arc  $a \rightarrow v$  outgoing an AFC node  $a \in \mathcal{A}$ , we associate the atomic function  $g^{(a \rightarrow v)}(\cdot)$ , which takes the  $|\mathcal{V}^{(a)}|$  length- $L$  incoming vectors  $\mathbf{x}^{(u \rightarrow a)}$ ,  $u \in \mathcal{V}^{(a)}$ , and produces the length- $L$  outgoing vector  $\mathbf{x}^{(a \rightarrow v)}$ , i.e.:

$$\mathbf{x}^{(a \rightarrow v)} = g^{(a \rightarrow v)} \left( \{ \mathbf{x}^{(u \rightarrow a)} \}_{u \in \mathcal{V}^{(a)}} \right).$$

Similarly, with each arc  $s \rightarrow v$  outgoing a source node  $s \in \mathcal{S}$ , the atomic function  $g^{(s \rightarrow v)}(\cdot)$  takes the  $|\mathcal{V}^{(s)}|$  length- $L$  incoming vectors  $\mathbf{x}^{(u \rightarrow s)}$ ,  $u \in \mathcal{V}^{(s)}$ , as well as the  $K$  generated symbols  $\sigma_s = (\sigma_s[1], \dots, \sigma_s[K])$ , and produces the length- $L$  outgoing vector  $\mathbf{x}^{(s \rightarrow v)}$ , i.e.:

$$\mathbf{x}^{(s \rightarrow v)} = g^{(s \rightarrow v)} \left( \{ \mathbf{x}^{(u \rightarrow s)} \}_{u \in \mathcal{V}^{(s)}}; \sigma_s \right).$$

(Note that we consider here the most general case in which a source node does not have to lie on the “bottom-most” level of the network, i.e., it can also have some incoming edges.) We refer here to both  $g^{(a \rightarrow v)}$ 's and  $g^{(s \rightarrow v)}$ 's as *encoding functions*.

Finally, a destination node  $d \in \mathcal{D}$  takes its  $|\mathcal{V}^{(d)}|$  incoming length- $L$  messages and performs decoding, i.e., it produces the vector of function evaluation estimates  $\hat{\mathbf{f}}^{(d)} = (\hat{f}^{(d)}[1], \dots, \hat{f}^{(d)}[K])$ , as follows:

$$\hat{\mathbf{f}}^{(d)} = \Psi^{(d)} \left( \{ \mathbf{x}^{(u \rightarrow d)} \}_{u \in \mathcal{V}^{(d)}} \right),$$

where  $\Psi^{(d)}(\cdot)$  is the destination node  $d$ 's function. Note that  $\Psi^{(d)}(\cdot)$  recovers back the  $K$ -dimensional vector from the  $L$ -dimensional incoming quantities (where  $L > K$ ), and it is therefore referred to as a decoding function.

We say that the destination  $d \in \mathcal{D}$  *computes* the function  $f : \mathbb{A}^N \rightarrow \mathbb{B}$ , if for every generation  $k \in \{1, \dots, K\}$ , it holds that:

$$\hat{f}^{(d)}[k] = f(\sigma_1[k], \dots, \sigma_N[k]).$$

Further, we say that the problem of computing  $f$  is *solvable* if there exist atomic functions  $g^{(s \rightarrow v)}(\cdot)$ ,  $g^{(a \rightarrow v)}(\cdot)$  across all arcs in  $\mathcal{E}$  and decoding functions  $\Psi^{(d)}(\cdot)$ ,  $d = 1, \dots, R$ , such that  $f$  is computed at all destinations  $d \in \mathcal{D}$  (that is, their corresponding composition computes  $f$  at all destinations).

**Connection to network coding and beyond.** The reader can observe that the problem of network coding [17] is a special case of the function computation problem with  $L = K = 1$ , where the target function  $f$  is an identity function:  $f(\sigma_1, \sigma_2, \dots, \sigma_N) = (\sigma_1, \sigma_2, \dots, \sigma_N)$ . In particular, in linear network coding, the alphabet  $\mathbb{A}$  is taken as a finite field  $\mathbb{F}$ , the function alphabet  $\mathbb{B}$  is  $\mathbb{F}^N$ , and all encoding functions  $g^{(a \rightarrow v)}$ ,  $g^{(s \rightarrow v)}$  and decoding functions  $\Psi^{(d)}$  are linear mappings over  $\mathbb{F}$ . The case of linear network coding is relatively well understood. In particular, it is known that the problem of computing  $f$  is solvable if and only if each of the minimum cuts between all the sources and any destination has capacity of at least  $N$  [60]. In Subsection VI-A, we provide further details on this special case.

For non-linear network coding, the universal criteria for network coding problem solvability are not fully understood. It is known, for example, that for the case where each sink requests a subset of the original messages, there exist networks, which are not solvable by using linear functions  $g^{(a \rightarrow v)}(\cdot)$ ,  $g^{(s \rightarrow v)}(\cdot)$  and  $\Psi^{(d)}(\cdot)$ , yet they can be solved by using non-linear functions (see, for example, [19]).

In order to understand the fundamental limits on solvability of the general function computation problem, the authors of [15] define what they term the computing capacity of a network as follows:

$$\mathcal{C}(\mathcal{G}, f) = \sup \left\{ \frac{K}{L} : \text{computing } f \text{ in } \mathcal{G} \text{ is solvable} \right\}. \quad (3)$$

They derive a general min-cut type upper bound on the computing capacity, as well as a number of more specific lower bounds. In particular, special classes of functions, such as symmetric functions, divisible functions and exponential functions, are considered therein (see [19] for more detail). It should be mentioned that the considered classes of functions are rather restricted, and that they possess various symmetry properties. The problem turns out to be very difficult, however, for more general, i.e., less restricted, classes of functions.

Another related work is [18], where a set-up with linear functions  $g^{(a \rightarrow v)}(\cdot)$ ,  $g^{(s \rightarrow v)}(\cdot)$  and  $\Psi^{(d)}(\cdot)$  and general linear target function  $f$  is considered. The authors are able to characterize some classes of functions, for which the cut-set bound gives sufficient condition for solvability, and for which it does not.

**Other results.** In [6], the function computation rate is defined and lower bounds on such rate are obtained for various simple classes of functions. Recently, in [65], information-theoretic bounds on the function computation rate were ob-



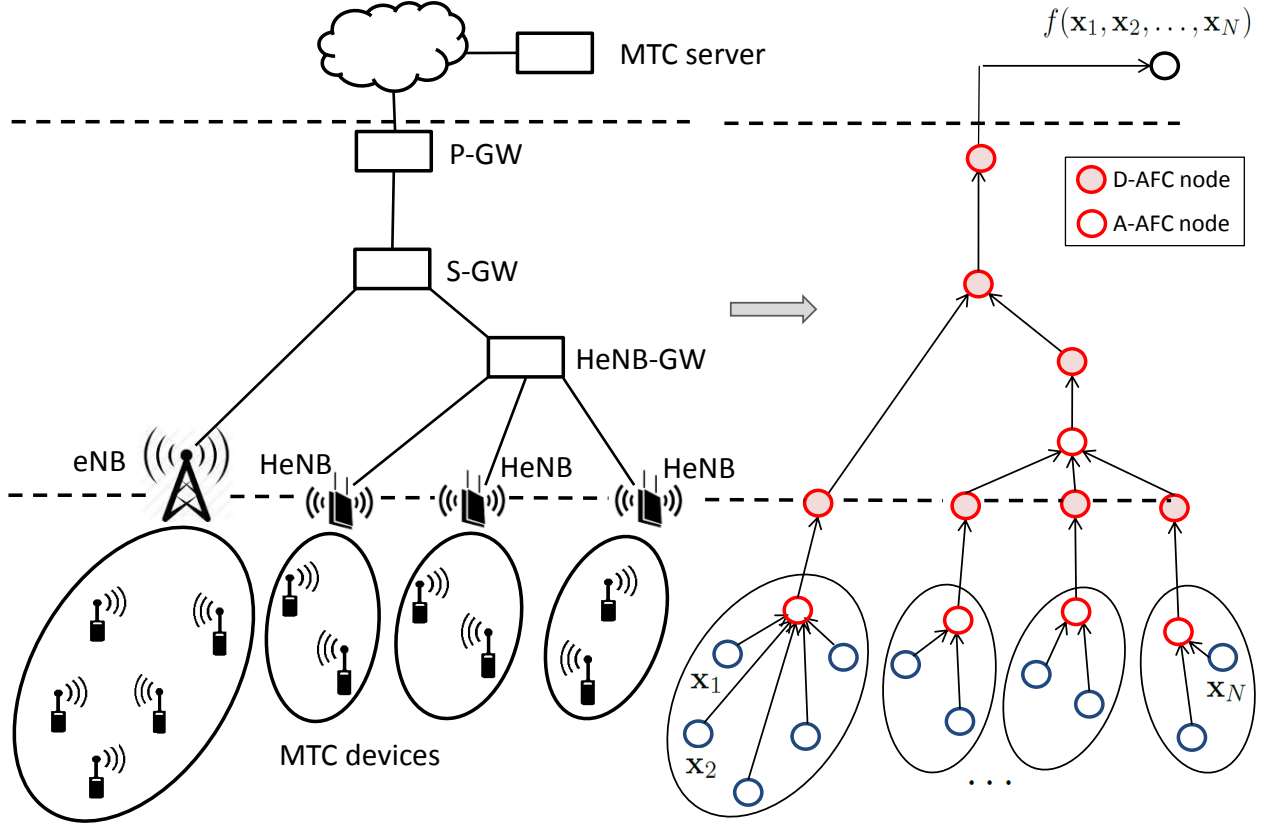


Fig. 6. Mapping between MTC network and NFC graph.

tained for a special case, when network is a directed rooted tree, and the set of sinks contains only the root of the tree, and the source symbols satisfy a certain Markov criterion.

A number of works study computation of sum in the network. It is shown in [63] that if each sink requests a sum of the source symbols, the linear coding may not be sufficient in networks where non-linear coding can be sufficient. Other related works include [59], [61], [62], [64], [66].

There is a significant number of works related to *secure* function computation available in the literature. However, usually, the main focus of these works is different. We leave that topic outside of the scope of this paper.

**Challenges and research directions.** Research on network function computation is still in its infancy and general theoretic foundations are yet to be developed. Here, we identify several challenges with network function computation relevant for Condense architecture. First, it is important to consider the issue of solvability when the encoding and decoding functions  $g^{(a \rightarrow v)}(\cdot)$ ,  $g^{(s \rightarrow v)}(\cdot)$  and  $\Psi^{(d)}(\cdot)$  are restricted to certain classes dictated by the underlying physical domain. For instance, A-AFC modules operating in the wireless domain are currently restricted to a certain class of functions (see Subsection IV-A), while, clearly, D-AFCs operating in the digital domain have significantly more powerful capabilities. Second, it interesting to study NFC in simpler cases, when the network topology is restricted to special classes of graphs, for example rooted trees, directed forests, rings, and others. Third, under the above defined constraints on the AFC capabilities, the

question that arises is how well we can approximate a desired function, even if solvability is impossible. Finally, practical and efficient ways for *actual constructions* of  $g^{(a \rightarrow v)}(\cdot)$ ,  $g^{(s \rightarrow v)}(\cdot)$  and  $\Psi^{(d)}(\cdot)$ , as opposed to existence-type results, are fundamental for Condense implementation.

### B. Implementation Aspects of NFC Layer

In practical terms, the NFC layer should deal with control and management tasks of establishing and maintaining an NFC graph of AFC modules for a given service request, as sketched in Fig. 6. In our vision, the main control modules that define the NFC layer functionality are: the function processor (FP) and the topology processor (TP) (see Fig. 2). Both modules can be seamlessly integrated in the SDN/NFV architecture.

The TP module organizes the MTC data flows and sends configuration instructions via the SDN control plane. In abstract terms, for all nodes in a directed rooted tree (or directed acyclic graph), TP needs to provide the set of child nodes  $\{\mathcal{V}^{(v)}\}_{v \in \mathcal{V}}$  from which to accept MTC data flows, and to identify the exact MTC data flows that will be filtered for each output flow, if there are multiple output flows.

Based on the MTC server application requests and the configured topology, FP processes the global function request, and, based on the available library of AFC modules, it generates the set of atomic functions to be used:  $\{g^{(a)}\}_{a \in \mathcal{A}}$ . Note that, as described before, AFC modules may be: i) A-AFC modules, ii) hardware-based D-AFC modules, and ii) software-based D-AFC modules. A-AFC modules (e.g., PLNC module)

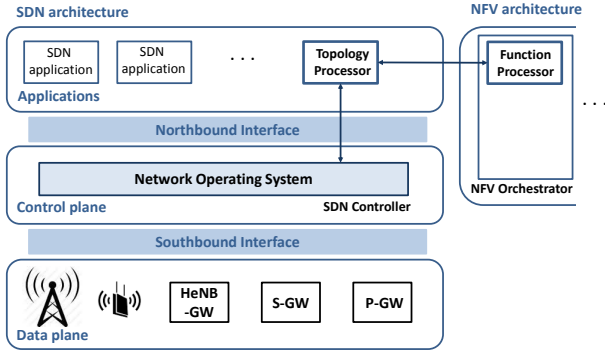


Fig. 7. NFC layer modules within SDN/NFV architecture.

need more complex instantiation control as they spread over several physical nodes and involve configuration of input and output interfaces and pre/post-processing functions (Sec. IVA). Hardware-based D-AFC modules (e.g., internal/external FPGA modules within or attached to network elements) require SDN-based control of MTC data flows that will filter selected flows and direct them through the D-AFC module. Finally, the most flexible case of software-based D-AFC modules (e.g., software-based modules in virtual machines running over the virtualized hardware in network elements or external servers) is a library of AFC implementations where each atomic function from the library can be remotely instantiated via the NFV control. Overall, FP needs to know the list of available AFC resources in the entire NFC network in order to optimize the set of instantiated AFC modules.

In terms of realization, we use the standard proposal for NFV/SDN complementary coexistence [30] where the FP module can be implemented as an NFV architecture block called the NFV orchestrator. The TP module can be implemented as an SDN application. Besides communicating directly, both FP and TP modules, observed as SDN applications, approach the SDN control plane via the SDN northbound interface. Based on the FP/TP inputs, the SDN control plane will configure physical devices (network nodes) via the southbound interface. This NFV/SDN based control of Condense is illustrated in Fig. 7.

## VI. APPLICATION LAYER

In this section, we present three examples of applications at the application layer of the Condense architecture: data recovery through network coding, minimizing population risk via a stochastic gradient method, and binary classification via neural networks<sup>2</sup>. The purpose of these examples is two-fold. First, they demonstrate that a wide range of applications can be handled via the Condense architecture. Second, they show that Condense is compatible with widely-adopted concepts in learning and communications, such as random linear network coding, stochastic gradient methods, and neural networks.

<sup>2</sup>Strictly speaking, learning a neural network can be considered a special case of a stochastic gradient method (with a non-convex loss function). We present it here as a distinct subsection as we consider the implementation where the neural network weight parameters are distributed across the Condense network.

The three examples are also complementary from the perspective of the workload required by the FP module. With the first example (data recovery via network coding), the function of interest is decomposed into mutually uncoordinated atomic (random) linear functions, and hence no central intervention by the function controller is required, nor is the inter-AFC modules coordination needed as long as atomic functions are concerned. With the third example (binary classification via neural networks), the desired network-wide function is realized through a distributed coordination of the involved AFC modules. Finally, with the second example (minimizing population risk via a stochastic gradient method), the most generic case requires the intervention of the central FP module, in order that the desired network-wide function be decomposed and computed.

### A. Data recovery through network coding

A special case of an application task with the Condense architecture is to deliver the *raw data* to the data center of interest. This corresponds to a trivial, identity function over the input data as a goal of the overall network function computation. However, this is not achieved through simply forwarding the raw data to the data center, but through the usage of network coding. In other words, atomic functions are not identity functions but random linear combinations over the input data. While such solution may not reduce the total communication cost with respect to the conventional (forwarding) solution, this solution is significantly more flexible, robust and reliable, e.g., [33], [34]. As recently noted, it can be flexibly implemented within the context of network coded cloud storage [67].

We follow the standard presentation of linear network coding, e.g., [68], adapting it to our setting. For ease of presentation, we assume here that graph  $\mathcal{G}$  is a directed rooted tree. Therein, the destination node is the root of the tree. Suppose that each of the  $N$  available MTC devices has a packet  $\mathbf{x}_s$  consisting of  $L$  symbols, each symbol belonging to a finite field  $\mathbb{F}$ . We adopt the finite field framework as it is typical with network coding. The goal is to deliver the whole packet vector  $\mathbf{x} = (\mathbf{x}_1, \dots, \mathbf{x}_N)$  to the data center (the destination node  $d$ ). With Condense, this is achieved as follows. Each atomic node  $a$  generates the message pair  $(\mathbf{x}^{(a)}, \mathbf{c}^{(a)})$  (to be sent to the parent node) based on the received messages from its child nodes  $(\mathbf{x}^{(b)}, \mathbf{c}^{(b)})$ , where  $b \in \mathcal{V}^{(a)}$ , and we recall that  $\mathcal{V}^{(a)}$  is the set of child nodes of node  $a$ . As we will see, the quantity  $\mathbf{x}^{(a)} \in \mathbb{F}^L$  is by construction a linear combination of the (subset of) MTC devices' packets  $\mathbf{x}_s \in \mathbb{F}^L$ ,  $s = 1, \dots, N$ . The quantity  $\mathbf{c}^{(a)} = (c^{(a)}[1], \dots, c^{(a)}[N]) \in \mathbb{F}^N$  stacks the corresponding weighting (or coding) coefficients; that is:

$$\mathbf{x}^{(a)} = \sum_{s=1}^N c^{(a)}[s] \mathbf{x}_s. \quad (4)$$

Now, having received  $(\mathbf{x}^{(b)}, \mathbf{c}^{(b)})$ ,  $b \in \mathcal{V}^{(a)}$ , node  $a$  computes  $\mathbf{x}^{(a)}$  using random linear network coding approach. It first generates new random (local) coding coefficients  $e^{(a)}[b] \in \mathbb{F}$ ,  $b \in \mathcal{V}^{(a)}$ , uniformly from  $\mathbb{F}$ , and independently of the received

messages. Then, it forms  $x^{(a)}$  as:

$$x^{(a)} = \sum_{b \in \mathcal{V}^{(a)}} e^{(a)}[b] \mathbf{x}^{(b)}.$$

Once  $\mathbf{x}^{(a)}$  has been computed, node  $a$  also has to compute the global coding coefficients  $\mathbf{c}^{(a)}$  with respect to the MTC packets  $\mathbf{x}_s$ ,  $s = 1, \dots, N$ , as per (4). It can be shown that:

$$c^{(a)}[s] = \sum_{b \in \mathcal{V}^{(a)}} e^{(a)}[b] c^{(b)}[s], \quad s = 1, \dots, N.$$

For the end-leaf (MTC device) nodes  $s$ , we clearly have that  $c^{(s)}[s] = 1$ , and  $c^{(s)}[u] = 0$ ,  $u \neq s$ . Once the destination (root) node  $d$  receives all its incoming messages, it has available a random linear combination of the MTC's packets  $\mathbf{x}_1, \dots, \mathbf{x}_N$ :

$$\mathbf{x}^{(d),1} = \sum_{s=1}^N c^{(d),1}[s] \mathbf{x}_s,$$

and the corresponding global coding coefficients vector  $\mathbf{c}^{(d),1} = (c^{(d),1}[1], \dots, c^{(d),1}[N])$ . Afterwards, the whole process described above is repeated sequentially  $N' - 1$  times, such that the data center obtains  $N' - 1$  additional pairs  $(\mathbf{x}^{(d),k}, \mathbf{c}^{(d),k})$ ,  $k = 2, \dots, N'$ . It can be shown that, as long as  $N'$  is slightly larger than  $N$ , MTC data vector  $\mathbf{x} = (\mathbf{x}_1, \dots, \mathbf{x}_N)$  can be recovered with high probability through solving the linear system of equations with unknowns  $\mathbf{x}_s$ :

$$\mathbf{x}^{(d),k} = \sum_{s=1}^N c^{(d),k}[s] \mathbf{x}_s, \quad k = 1, \dots, N'.$$

Note that, for this application example, each atomic function is *linear*. Moreover, there is no requirement on the *coordination* of the atomic functions which correspond to different atomic nodes, as they are generated randomly and mutually independently [69]. Hence, this application does not require a centralized control by the FP module.

Finally, when certain a priori knowledge on  $\mathbf{x}$  is available (e.g., sparsity, i.e., many of the packets  $\mathbf{x}_s$  are the zero  $L$ -tuples of symbols from  $\mathbb{F}$ ), then the recovery probability close to one can be achieved even when the number of linear combinations  $N'$  at the MTC server is significantly smaller than  $N$ . Omitting details, this can be in principle achieved using the theories of compressed sensing and sparse recovery, e.g., [70], [71].

## B. Statistical estimation and learning

A dominant trend in current machine learning research are algorithms that scale to large datasets and are amenable to modern distributed processing systems. Machine learning systems are widely deployed in architectures with a large number of processing units at different physical locations and communication is becoming a resource that is taking the center stage in the algorithm design considerations [72]–[76].

Typically, the task of interest (parameter estimation, prediction, etc.) is performed through solving an optimization problem of minimizing a risk function [77]. In most widely used models of interest, which include logistic regression and neural networks, this optimization needs to be performed numerically using gradient descent methods and is simply

based on successive computation of the gradient of the loss function of interest. In large datasets (of size  $T$ ), obtaining the full gradient comes with a prohibitive computational (a linear computational cost in  $T$  per iteration of gradient descent cannot be afforded) as well as a prohibitive communication cost (due to the need to access *all* training examples even though they may be, and typically are, stored at different physical locations). For these reasons, stochastic gradient methods are the norm – they typically access only a small number of data points at a time, giving an unbiased estimate to the gradient of the loss function needed to update the parameter values – and have enjoyed tremendous popularity and success in practice [78]–[81].

Most existing works assume that the data has already been collected and transmitted through the communication architecture and is available at request. That is, typically the data is first transmitted in its raw form from the MTC devices to the data center, and only afterwards a learning (optimization) algorithm is executed.

In contrast, the Condense architecture integrates the learning task into the communication infrastructure. That is, a learning task is seen as a sequence of oracle calls to a certain network function computation, and the role of the NFC layer is to provide these function computations at the data center's processing unit (destination node) and only the computed value (e.g., of the gradient to the loss function) is being communicated. This way, Condense will generically embed various learning algorithms into the actual 3GPP MTC communication infrastructure.

We now dive into more details and exemplify learning over the proposed Condense system with the estimation of an unknown parameter vector  $\mathbf{w}^* \in \mathbb{R}^Q$  through the minimization of population risk. Specifically, we consider stochastic gradient-type methods to minimize the risk.

To begin, consider a directed rooted tree NFC graph  $\mathcal{G}$ , and assume that there are  $N$  MTC devices which generate samples  $\mathbf{x}_t \in \mathbb{R}^{L \cdot N}$  over time instants  $t = 0, 1, 2, \dots$ , drawn i.i.d. in time from a distribution  $\mathbf{X} = (\mathbf{X}_1, \dots, \mathbf{X}_N) \sim \mathbb{P}$ , defined over  $\mathbb{R}^{L \cdot N}$ . Here,  $\mathbf{x}_t = (\mathbf{x}_{1,t}, \dots, \mathbf{x}_{N,t})$ , where  $\mathbf{x}_{s,t} \in \mathbb{R}^L$  is a sample of  $\mathbf{X}_s$  generated by the MTC device  $s$ . The goal is to learn the parameter vector  $\mathbf{w}^* \in \mathbb{R}^Q$  that minimizes the population risk:

$$\mathbb{E}_{\mathbf{X}} [\phi(\mathbf{w}; \mathbf{X})], \quad (5)$$

where expectation is well-defined for each  $\mathbf{w} \in \mathbb{R}^Q$ , and, for each  $\mathbf{x} \in \mathbb{R}^{L \cdot N}$ , function  $\phi(\cdot; \mathbf{x}) : \mathbb{R}^Q \rightarrow \mathbb{R}$  is differentiable. In the rest of this subsection, we specialize the approach on a single but illustrative example of Consensus; more elaborate examples such as logistic regression are relevant but not included here for brevity.

**Example: Consensus – computing the global average; e.g., [82]–[84].** When  $w \in \mathbb{R}$ ,  $\mathbf{x} = (x_1, \dots, x_N) \in \mathbb{R}^N$  (each MTC device generates scalar data), and  $\phi(w, \mathbf{x}) = \frac{1}{N} \sum_{s=1}^N (x_s - w)^2$ , then solving (5) corresponds to finding  $\frac{1}{N} \sum_{s=1}^N \mathbb{E}[X_s]$ . E.g., when MTC devices are pollution sensors at different locations in a city, this corresponds to finding the city-wide average pollution.

**Conventional 3GPP MTC solution.** Consider first the conventional 3GPP MTC system, where samples  $\mathbf{x}_t$ ,  $t = 0, 1, 2, \dots$ , arrive (through the communication layer) to a processing unit at the data center at time instants  $t = 0, 1, 2, \dots$  in their raw form. (We ignore here the communication delays.) Upon reception of each new sample  $\mathbf{x}_t$ , the processing unit (destination node  $d$ ) performs a stochastic gradient update to improve its estimate  $\mathbf{w}^{(t)} \in \mathbb{R}^Q$  of  $w^*$ :

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta_t \nabla \phi(\mathbf{w}^{(t)}; \mathbf{x}_t), \quad (6)$$

where  $\nabla \phi(\mathbf{w}^{(t)}; \mathbf{x}_t)$  is the gradient of  $\phi(\cdot; \mathbf{x}_t)$  at  $\mathbf{w}^{(t)}$ , and  $\eta_t$  is the step-size (learning rate). For the consensus example with learning rate  $\eta_t = 1/(t+1)$ , it can be shown that update (6) takes the particularly simple form:

$$w^{(t+1)} = \frac{t}{t+1} w^{(t)} + \frac{1}{t+1} \left( \frac{1}{N} \sum_{s=1}^N x_{s,t} \right). \quad (7)$$

Note that, with the conventional 3GPP MTC architecture, data samples  $\mathbf{x}_t$  are transmitted to the destination node  $d$  for processing in their entirety, i.e., the communication infrastructure acts only as a routing network (forwarder) of the data.

**Condense solution.** In contrast with the conventional solution, with the Condense architecture the raw data sample  $\mathbf{x}_t$  is not transmitted to the data center and is hence not available at the corresponding processing unit. Instead, update (6) is implemented as follows. Given the current estimate  $\mathbf{w}^{(t)}$ , the processing unit (destination node  $d$ ) defines function  $f_t(\cdot) := \nabla \phi(\mathbf{w}^{(t)}; \cdot)$ . Subsequently, it sends the request to the function processor to perform the decomposition of  $f_t(\cdot)$  over the NFC layer. The function processor performs the required decomposition of  $f_t(\cdot)$  into atomic functions and remotely installs the corresponding obtained atomic function at each atomic node (eNB, HeNB, etc.) of the topology.<sup>3</sup> Once the required atomic functions are ready, the sample  $\mathbf{x}_t$  starts travelling up the graph  $\mathcal{G}$ , and upon the completion of evaluation of all intermediate atomic functions, the value  $f_t(\mathbf{x}_t)$  becomes available at the data center. This in turn means that the processing unit can finalize update (6). Specifically, with the consensus example in (7), function  $f_t(\cdot)$  takes a particularly simple form of the average:  $f_t(x) = f(x) = \frac{1}{N} \sum_{s=1}^N x_s$ , and it is independent of  $w^{(t)}$  and of  $t$ . There exist many simple and efficient methods to decompose<sup>4</sup> the computation of the average, e.g., [85], and hence algorithm (7) can be implemented very efficiently within the Condense architecture.

**Challenges, insights and research directions.** We close this subsection by discussing several challenges which arise when embedding learning algorithms in the Condense architecture. Such challenges are manifold but are nonetheless

already a reality in machine learning practice. First, data arrives in an asynchronous, delayed, and irregular fashion, and it is often noisy. Condense actually embraces this reality and puts the learning task at the center stage: the desired function of the data is of interest, not the data itself. Secondly, it is often the case that, depending on the infrastructure, interface and functionality constraints of the network computation layer, approximations of the desired function computations (as opposed to exact computations) will need to be employed. For instance, function  $f_t(\cdot) := \nabla \phi(\mathbf{w}^{(t)}; \cdot)$  in the example above may only be computable approximately in general. The quality of such an approximation leads to trading-off statistical efficiency of the learning procedure with the accuracy of the network function computation, and the analyses of such trade-offs will be an important research topic. Finally, from a more practical perspective, an important issue is to ensure interoperability with the existing distributed processing paradigms (e.g., Graphlab [86] and Hadoop [87]).

### C. Neural networks

With modern large scale applications of neural networks, the number of parameters to be learned (neuron's weights) can be excessively large, like, e.g., with deep neural networks [21]. As such, storage of the parameters themselves should be distributed, and their updates also include a large communication cost that needs to be managed [20]. However, neural networks can be naturally embedded into the Condense architecture – somewhat similarly to the related work on distributed training for deep learning [23] – as detailed next.

Specifically, consider the example of binary classification of the MTC devices' generated data. At each time instant  $t$ ,  $N$  MTC devices generate data vector  $\mathbf{x}_t = (\mathbf{x}_{1,t}, \dots, \mathbf{x}_{N,t}) \in \mathbb{R}^{N \cdot L}$ , where each device generates an  $L$ -dimensional vector  $\mathbf{x}_{i,t}$ . Each data vector  $\mathbf{x}_t$  is associated with its class label  $y_t \in \{-1, 1\}$ . A binary classifier  $F: \mathbb{R}^{N \cdot L} \rightarrow \{-1, 1\}$  takes a data sample  $\mathbf{x}_t$  and generates an estimate  $F(\mathbf{x}_t)$  of its class label  $y_t$ . Classifier  $F$  is “learned” from the available training data  $(\mathbf{x}_t, y_t)$ ,  $t = 0, 1, \dots, T$ , where  $T$  is the *learning period*. In other words, once the learning period is completed and  $F$  is learned, then the *prediction period* is initiated, and for each new data sample  $\mathbf{x}_t$ ,  $t > T$ , classifier  $F$  generates an estimate of  $y_t$ . For example,  $\mathbf{x}_t$  can correspond to measurements of pressure, temperature, vibration, acoustic, and other sensors in a large industrial plant within a time period  $t$ ;  $y_t = 1$  can correspond to the “nominal” plant operation, while  $y_t = -1$  to the “non-nominal” operation, defined for example as the operation where energy efficiency or greenness standards are not fully satisfied.

We consider neural network-based classifiers  $F$  embedded in the Condense architecture. Therein, the classifier function  $F$  is a composition of the neuron functions associated with each AFC module (node). We consider a Condense rooted tree graph  $\mathcal{G}$  with  $N$  sources and one destination node  $d$ , however, here we assume  $\mathcal{G}$  is *undirected* as we will need to pass messages upwards and downwards. For convenience, as it is common with neural networks, we organize all nodes in  $\mathcal{G}$  (source, atomic, and the destination node) in levels

<sup>3</sup>We assume that performing decomposition of  $f_t(\cdot)$  and the installation of the atomic functions across the AFC layer is completed prior to the initiation of flow of sample  $\mathbf{x}_t$  “onwards” through the NFC topology. In other words, the time required for the latter process is sufficiently smaller than the time intervals of generation of data samples  $\mathbf{x}_t$ .

<sup>4</sup>Strictly speaking,  $f_t(x) = \frac{1}{N} \sum_{s=1}^N x_s$  is not defined for the consensus example here as the gradient of  $\phi(w, \cdot)$  at  $x$ , but it is defined as the additive term in (6) which is dependent upon  $\mathbf{x}_t$ . The gradient actually equals  $\frac{1}{N} \sum_{s=1}^N (w - x_s)$ ; applying (5) to this gradient form with  $\eta_t = 1/(t+1)$  yields (6).

$\ell = 1, 2, \dots, \mathcal{L}$ , such that the leaves of nodes at the first level ( $\ell = 1$ ) are the MTC devices (sources in  $\mathcal{S}$ ), while the data center's processing unit (the destination node  $d$ ) corresponds to  $\ell = \mathcal{L}$ . Then, all nodes (sources, atomic nodes, and the destination node) are indexed through the index pair  $(\ell, m)$ , where  $\ell = 0, 1, \dots, \mathcal{L}$  is the level number and  $m$  is the order number of a node within its own level,  $m = 1, \dots, \mathcal{N}_\ell$ . Here,  $\mathcal{N}_\ell$  denotes the number of nodes at the  $\ell$ -th level.

**Prediction.** We first consider the *prediction period*  $t > T$ , assuming that the learning period is completed. This corresponds to actually executing the application task of classification, through evaluating the network function  $F$  at a data sample  $\mathbf{x}_t$ . (As we will see ahead, the learning period corresponds to *learning* function  $F$ , which essentially parallels the task of how a desired network function is decomposed across the AFC modules into the appropriate atomic functions.) Each node  $(\ell, m)$  is assigned a weight vector  $\mathbf{w}^{(\ell, m)}$  (obtained within the learning period), whose length equals the number of its associated leaf nodes. Denote by  $x_t^{(\ell, m)}$  the output (also referred to as activity) of node  $(\ell, m)$  associated with the data sample  $\mathbf{x}_t$ ,  $t > T$ , to be computed based on the incoming activities  $\mathbf{x}_t^{(\ell-1, q)}$  from the adjacent lower level nodes  $(\ell-1, q)$ . Also, denote by  $\mathbf{x}_t^{(\ell-1)}$  the vector that stacks all the  $\mathbf{x}_t^{(\ell-1, q)}$ 's at the level  $\ell-1$ . Then,  $x_t^{(\ell, m)}$  is calculated by:

$$x_t^{(\ell, m)} = \mathcal{U} \left( (\mathbf{w}^{(\ell, m)})^\top \mathbf{x}_t^{(\ell-1)} \right), \quad (8)$$

where  $z \in \mathbb{R} \mapsto \mathcal{U}(z) = \frac{1}{1+\exp(-z)}$  is the logistic unit function. Therefore, with neural networks, the atomic function  $g^{(\ell, m)}(\cdot)$  associated with each AFC module (node)  $(\ell, m)$  is a composition of 1) the linear map parameterized with its weight vector  $\mathbf{w}^{(\ell, m)}$ ; and 2) the logistic unit function.

**Learning.** The learning period corresponds to learning function  $F$ , i.e., learning the weight vectors  $\mathbf{w}^{(\ell, m)}$  of each AFC module. Differently from the example of minimizing a generic population risk in Subsection VI-B, here learning  $F$  (learning atomic functions of AFC modules) can be done in a distributed way, without the involvement of the FP module. The learning is distributed in the sense that it involves passing messages in the “upward” direction (from the MTC devices towards the data center) and the “downward” direction (from the data center towards the MTC devices) along the Condense architecture (graph  $\mathcal{G}$ ).

Specifically, we assume that weight vectors  $\mathbf{w}^{(\ell, m)}$  are learned by minimizing the log-loss  $J(\{\mathbf{w}^{(\ell, m)}\})$  via a stochastic gradient descent (back-propagation) algorithm, wherein one upward/downward pass corresponds to a single training data sample  $\mathbf{x}_t$ ,  $t \leq T$ . We now proceed with detailing both the upward and the downward pass [22]. We assume that, before initiating the pass,  $\mathbf{x}_t$  is available at the bottom-most layer (MTC devices), while label  $y_t$  is available at the data center (destination node  $d$ ). This is reasonable to assume as the label's data size per  $t$  is insignificant (here it is just one bit) and can be delivered to the data center, e.g., by forwarding (conventional) means through the 3GPP MTC system.

**Upward pass.** Each node  $(\ell, m)$  computes the gradient of its activity with respect to its weights as well as with respect

to the incoming activities:

$$\frac{\partial x_t^{(\ell, m)}}{\partial \mathbf{w}^{(\ell, m)}} = x_t^{(\ell, m)} \left( 1 - x_t^{(\ell, m)} \right) \mathbf{x}_t^{(\ell-1)}.$$

$$\frac{\partial x_t^{(\ell, m)}}{\partial \mathbf{x}_t^{(\ell-1)}} = x_t^{(\ell, m)} \left( 1 - x_t^{(\ell, m)} \right) \mathbf{w}^{(\ell, m)}.$$

At this point, node  $(\ell, m)$  stores tuple  $\left( t, \frac{\partial x_t^{(\ell, m)}}{\partial \mathbf{w}^{(\ell, m)}}, \frac{\partial x_t^{(\ell, m)}}{\partial \mathbf{x}_t^{(\ell-1)}} \right)$  (these are local gradients, needed for weight update in the downward pass.).

**Downward pass.** Label  $y_t$  has been received at the data center's processing node. Now gradients of loss function  $J$  are backpropagated. Having obtained  $\frac{\partial J}{\partial x_t^{(\mathcal{L})}}$ , each node  $(\ell, m)$  sends to its lower layer neighbour  $(\ell-1, k)$  the message consisting of  $(t, \delta_t^\ell(m \rightarrow k))$  (which we refer to here as gradient contribution), where

$$\begin{aligned} \delta_t^\ell(m \rightarrow k) &= \frac{\partial J}{\partial x_t^{(\ell, m)}} \frac{\partial x_t^{(\ell, m)}}{\partial \mathbf{x}_t^{(\ell-1, k)}} \\ &= \frac{\partial J}{\partial x_t^{(\ell, m)}} x_t^{(\ell, m)} \left( 1 - x_t^{(\ell, m)} \right) \mathbf{x}_t^{(\ell-1)}. \end{aligned}$$

Node  $(\ell-1, k)$  now can compute

$$\frac{\partial J}{\partial x_t^{(\ell-1, k)}} = \sum_m \delta_t^\ell(m \rightarrow k).$$

This is instantiated at the top layer:

$$\frac{\partial J}{\partial x_t^{(\mathcal{L})}} = -\frac{y_t}{x_t^{(\mathcal{L})}} + \frac{1 - y_t}{1 - x_t^{(\mathcal{L})}}.$$

Moreover, after sending  $\delta_t^\ell(m \rightarrow k)$ , node  $(\ell, m)$  updates its weights with stochastic gradient update and step-size  $\eta_t$ :

$$\mathbf{w}^{(\ell, m)} \leftarrow \mathbf{w}^{(\ell, m)} - \eta_t \frac{\partial J}{\partial x_t^{(\ell, m)}} x_t^{(\ell, m)} \left( 1 - x_t^{(\ell, m)} \right) \mathbf{w}^{(\ell, m)},$$

and removes “local gradients” from the memory.

**Challenges, insights and research directions.** We close this subsection with several challenges and practical considerations which arise when embedding neural networks into the Condense architecture.

The first challenge is on implementing the required AFC modules (atomic functions) in the analog domain. These modules are typically linear combinations plus nonlinearities (sigmoids, rectified linear units). Secondly, even when they are implemented in the digital domain, an interesting question is to study the effects of propagation of the quantization error across the Condense architecture.

Next, network topology and busy nodes will dictate that not all nodes see the activities corresponding to the  $t$ -th example. This is just like the dropout method [24] which deliberately “switches off” neurons randomly during each learning stage. Dropout is a hugely successful method for learning regularization as it prevents overfitting by weight co-adaptation and demonstrates that the learning process can be inherently robust to the node failures, echoing the overall case against the learning and network layer separation, which



presumes all data to be available on request at all times. Moreover, many activities will not be sent to all the nodes in the upper layer. In this case,  $\frac{\partial x_t^{(\ell, m)}}{\partial w^{(\ell, m)}[k]} = 0$ , so weights will not be affected. In this case, corresponding gradients do not need to be stored, nor does the downward pass need to happen. More problematic is the situation in which upward pass has happened but downward pass fails at some point, i.e., some of the  $\delta_t^\ell(m \rightarrow k)$  are not received at  $(\ell - 1, k)$ . This injects additional noise to the gradient. Studying the effect of this noise is an interesting research topic.

We finally provide some insights on the communication and computational costs. Each AFC module (node) broadcasts one real number per training data example: its activity (together with data example index  $t$ ), in the upward pass, and one real number per example, per receiver: gradient contribution  $\delta_t^\ell(m \rightarrow k)$  (together with index  $t$ ), in the downward pass. Thus, each node broadcasts to upper layers and sends specific messages to specific nodes in bottom layers. Upward pass happens whenever a new input is obtained, while downward pass whenever a new output is obtained. Due to this asynchrony, gradient updates might be out of date – therefore, each node could purge local gradients for outdated examples.

## VII. OTHER IMPLEMENTATION ASPECTS

This Section briefly discusses some aspects of the Condense architecture not considered in earlier Sections.

**Size of NFC graph  $\mathcal{G}$ .** We first discuss a typical size of an NFC graph. Referring to Figure 6 and assuming a directed rooted tree graph, it can typically have depth (number of layers) around 5 – 7. Regarding the number of source nodes (MTC devices – lower most layer), it is estimated that the number of MTC devices per macro-cell eNB will be in the range of  $10^3 - 10^5$ . The number of small cells per macro cell is in the range of  $10^1 - 10^2$ , which makes the number of MTC devices per small cell approximately  $10^2 - 10^3$ . Assuming a  $30\text{km} \times 30\text{km}$  city area and a  $100\text{m} \times 100\text{m}$  coverage of a small cell, we can have a total of  $10^4 - 10^5$  small cells within a city-wide Condense network. Therefore, in a city-wide Condense network, we may have  $10^7 - 10^8$  MTC devices (number of nodes at the lower-most layer), and on the order of  $10^4 - 10^5$  nodes at the (base station) layer above. The number of nodes at the upper layers going further upwards is lower and is few tens or less. In summary, a typical city-wide Condense rooted tree network may have a total of  $10^7 - 10^8$  nodes, it has a large “width” and a moderate “depth”. This goes relatively well in line with the supporting theory; e.g., neural networks are considered deep with depths of order 7 or so, while arbitrary functions can be well-approximated even with shallow neural networks. Of course, the graph size can be virtually adjusted according to current application needs both horizontally (to adjust width) and vertically (to adjust depth) through implementing multiple (virtual) nodes within a single physical device.

**Synchronization.** We initially assess that synchronization may not be a major issue with realizing Condense. This is because, actually, synchronization is critical only with implementing analog atomic functions, e.g., within a single HeNB

module. Network-wide orchestration of atomic functions may be successfully achieved through the control mechanisms of SDN and NFV, as well as through the usage of *buffering* at the upper Condense layers (HeNB-GW, S-GW, and P-GW), to compensate for delays and asynchrony.

**Communication, computational, and storage costs.** We now discuss reductions of communication costs (per application task) of Condense with respect to the conventional (forwarding) 3GPP MTC solution. How much communications is saved depends largely on the application at hand. For very simple tasks (functions), like, e.g., computing maximum or global average, it is easy to see that the savings can be very high. In contrast, for forwarding (computing the identity function), the savings may not be achieved (but the reliability is improved through random linear network coding). Also, overall communication savings depend on the overhead incurred by the signalling from the topology and function processors to the AFC modules (in order to orchestrate the topology, perform function decomposition and instal the appropriate atomic functions, etc.) However, this overhead is projected to eventually become small, as, upon a significant development of the technology, atomic function libraries and NFC decompositions will be pre-installed. Further, it is clear that Condense requires additional storage and computational functionalities at network nodes, when compared with current 3GPP MTC systems. However, this is a reasonable assumption for the modules (eNBs, GWs, etc.) of 3GPP MTC systems due to upcoming trends in mobile edge computing [88].

**Data privacy and data loss.** Condense naturally improves upon privacy of IoT systems, as the data center (except when computing the identity function) does not receive the data in its raw form. Finally, if certain IoT-generated data has to be stored in a cloud data center in its raw form so as to ensure its long lifetime, Condense supports this functionality through identity functions. However, it is natural to expect that this request is, on average across all IoT data sources and all applications, only occasionally imposed, rendering significant communication savings overall.

Finally, Table I provides a summary of the proposed architecture. The table briefly indicates main points presented in this paper in terms of the advantages of the proposed architecture, relevant theoretical and implementation aspects, and main future research directions.

## VIII. CONCLUSIONS

In this paper, we proposed a novel architecture for knowledge acquisition of IoT-generated data within the 3GPP MTC (machine type communications) systems, which we refer to as Condense. The Condense architecture introduces a novel service within 3GPP MTC systems – computing linear and non-linear functions over the data generated by MTC devices. This service brings about the possibility that the underlying communication infrastructure communicates only the desired function of the MTC-generated data (as required by the given application at hand), and not the raw data in its entirety. This transformational approach has the potential to dramatically reduce the pressure on the 3GPP MTC communication infrastructure.

TABLE I  
SUMMARY OF CONDENSE ARCHITECTURE.

Features and pros	Theory and implementation		Future directions
<ul style="list-style-type: none"> <li>- Reconfigurable architecture;</li> <li>- Novel service of computing functions over MTC-data;</li> <li>- Three layers: atomic, network and application;</li> <li>- Two control elements: topology and function processor;</li> <li>- Can be integrated within 3GPP MTC architecture;</li> <li>- Can exploit theories of sensor fusion, network coding and computation and neural networks;</li> <li>- Can be customized for variety of MTC applications;</li> </ul>	<b>AFC layer</b>	<ul style="list-style-type: none"> <li>- Analog: wireless and optical domains;</li> <li>- Digital: FPGA/software;</li> <li>- Theory: Nomographic functions (analog wireless) and Reservoir computing (analog optical)</li> </ul>	<ul style="list-style-type: none"> <li>- Implementation challenges: channel estimation, timing and frequency offsets and quantization issues;</li> <li>- Development of standardized A-AFC and D-AFC modules;</li> </ul>
	<b>NFC layer</b>	<ul style="list-style-type: none"> <li>- Topology processor: NFV orchestrator;</li> <li>- Function processor: SDN application;</li> <li>- Theory: Network coding for computing, sensor fusion and neural networks</li> </ul>	<ul style="list-style-type: none"> <li>- Actual constructions of function decompositions;</li> <li>- Decomposability (solvability) under restricted function classes and network topologies;</li> <li>- Development of function and topology processor SDN/NFV modules;</li> </ul>
	<b>Application layer</b>	<ul style="list-style-type: none"> <li>- Implementation examples: RLNC, neural networks and stochastic gradient descent;</li> <li>- Theory: neural networks, statistical learning and prediction</li> </ul>	<ul style="list-style-type: none"> <li>- Asynchronous, delayed and irregular arrival of data;</li> <li>- Inexact network function computation;</li> <li>- Interoperability with existing data analytics platforms;</li> </ul>

The paper provides contributions along two main directions. First, from the architectural side, we describe in detail how the function computation service can be realized within 3GPP MTC systems. Second, from the theoretical side, we survey the relevant literature on the possibilities of realizing “atomic” functions in both analog and digital domains, as well as on the theories and techniques for function decomposition over networks, including the literature on sensor fusion, network coding for computing, and neural networks. The paper discusses challenges, provides insights, and identifies future research directions for implementing function computation and function decomposition within practical 3GPP MTC systems.

#### ACKNOWLEDGMENT

The authors thank the following researchers for valuable help in developing the Condense concept: J. Coon, R. Vicente, M. Greferath, O. Gnille, R. Freij-Hollanti, A. Vazquez Castro, V. Crnojevic, G. Chatzikostas, C. Mirasso, P. Colet, and M. C. Soriano. The authors would also like to acknowledge valuable support for collaboration through the EU COST IC 1104 Action.

#### REFERENCES

- [1] G. Rohling, “Facts and forecasts: Billions of things, trillions of dollars, pictures of the future,” available online: <http://www.siemens.com/innovation/en/home/pictures-of-the-future/digitalization-and-software/internet-of-things-facts-and-forecasts.html> (last accessed: 15.06.2016.)
- [2] T. Bjarin, “The Next Big Thing for Tech: The Internet of Everything, Time Magazine,” available online: <http://time.com/539/the-next-big-thing-for-tech-the-internet-of-everything/> (last accessed: 15.06.2016.)
- [3] <http://ec.europa.eu/digital-agenda/en/internet-things> (last accessed: 15.06.2016.)
- [4] <http://www.zdnet.com/topic/the-power-of-iot-and-big-data/> (last accessed: 15.06.2016.)
- [5] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, “Internet of Things (IoT): A vision, architectural elements, and future directions,” *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645-1660, 2013.
- [6] A. Giridhar, and P. R. Kumar, “Computing and communicating functions over sensor networks,” *IEEE Journal on Selected Areas in Communications*, vol. 23, no. 4, pp. 755-764, 2005.
- [7] A. Giridhar and P. R. Kumar, “Toward a theory of in-network computation in wireless sensor networks,” *IEEE Commun. Mag.*, vol. 44, no. 4, pp. 98-107, Apr. 2006.
- [8] T. Taleb, and A. Kunz, “Machine type communications in 3GPP networks: Potential, challenges, and solutions,” *IEEE Communications Magazine*, vol. 50, no. ), pp. 178-184, 2012.
- [9] The European Telecommunications Standards Institute. Network Functions Virtualisation (NFV); Architectural Framework. GS NFV 002 (V1.1.1), Oct. 2013.
- [10] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, “Network function virtualization: Challenges and opportunities for innovations,” *IEEE Communications Magazine*, vol. 53, no. 2, pp. 90-97, 2015.
- [11] B. Lantz, B. Heller, and M. McKeown, “A network in a laptop: rapid prototyping for software-defined networks,” *9th ACM SIGCOMM Workshop on Hot Topics in Networks*, pp. 19, ACM, 2010.
- [12] D. Kreutz, F. M. Ramos, P. Esteves Verissimo, C. Esteve Rothenberg, S. Azodolmolky, and S. Uhlig, “Software-defined networking: A comprehensive survey,” *Proc. IEEE*, vol. 103, no. 1, pp. 14-76, 2015.
- [13] M. Condoluci, G. Araniti, T. Mahmoodi, M. Dohler, “Enabling the IoT Machine Age with 5G: Machine-Type Multicast Services for Innovative Real-Time Applications,” *IEEE Access (transactions), Special Issue on IoT*, in press, 2016.
- [14] M.R. Palattella, M. Dohler, A. Grieco, G. Rizzo, J. Torsner, T. Engel, L. Ladid, “Internet of Things in the 5G Era: Enablers, Architecture and Business Models,” *IEEE Journal on Selected Areas in Communications*, in press, 2016.
- [15] R. Appuswamy, M. Franceschetti, N. Karamchandani, and K. Zeger, “Network coding for computing: cut-set bounds,” *IEEE Transactions on Information Theory*, vol. 57, no. 2, pp. 1015-1030, Feb. 2011.
- [16] H. Kowshik and P.R. Kumar, “Optimal function computation in directed and undirected graphs,” *IEEE Transactions on Information Theory*, 58(6), pp.3407-3418, June 2012.
- [17] R. Ahlswede, N. Cai, S.-Y.R. Li, and R.W. Yeung, “Network information flow,” *IEEE Transactions on Information Theory*, vol. 46, no. 4, pp. 1204-1216, Apr. 2000.
- [18] R. Appuswamy and M. Franceschetti, “Computing linear functions by linear coding over networks,” *IEEE Transactions on Information Theory*, vol. 60, no. 1, pp. 422-431, Jan. 2014.
- [19] R. Dougherty, C. Freiling, and K. Zeger, “Insufficiency of linear coding in network information flow,” *IEEE Transactions on Information Theory*, vol. 51, no. 8, pp. 2745-2759, Aug. 2005.
- [20] M. Li, D.G. Andersen, J.W. Park, A.J. Smola, A. Ahmed, V. Josifovski, J. Long, E.J. Shekita, and B.-Y. Su, “Scaling distributed machine learning with the parameter server,” *11th USENIX Symposium on Operating Systems Design and Implementation*, 2014.
- [21] Y. Bengio, Y. LeCun, and G. Hinton, “Deep learning,” *Nature* 521, pp. 436444, 2015.
- [22] B. D. Ripley, “Pattern recognition and neural networks,” Cambridge University Press, 1996.
- [23] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, Q. Le, M. Mao, M.

- Ranzato, A. Senior, P. Tucker, K. Yang, A. Ng, "Large scale distributed deep networks," *Advances in Neural Inf. Proc. Systems* 25, 2012.
- [24] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journ. Mach. Learn. Research*, vol. 15 (Jun), pp. 1929-1958, 2014.
- [25] 3GPP TS 22.368, "Service requirements for Machine-Type Communications (MTC)," V13.1.0, Dec. 2014.
- [26] H. Shariatmadari, R. Ratasuk, S. Iraj, A. Laya, T. Taleb, R. Jantti, and A. Ghosh, "Machine-type communications: current status and future perspectives toward 5G systems," *IEEE Communications Magazine*, vol. 53, no. 9, pp. 10-17, 2015.
- [27] L. E. Li, Z. M. Mao and J. Rexford, "Toward software-defined cellular networks," *European Workshop on Software Defined Networking (EWSN)*, pp. 7-12, October 2012.
- [28] <http://www.3gpp.org/DynaReport/32842.htm>
- [29] N. Zilberman, P. M. Watts, C. Rotsos, and A. W. Moore, "Reconfigurable network systems and software-defined networking," *Proceedings of the IEEE*, vol. 103, no. 7, pp. 1102-1124, 2015.
- [30] Y. Li, and M. Chen, "Software-defined network function virtualization: A survey," *IEEE Access*, 3, pp. 2542-2553, 2015.
- [31] S. Zhang, S. C. Liew, and P. P. Lam, "Hot topic: Physical-layer network coding," *ACM Annual Conference on Mobile Computing and Networking*, pp. 358-365, Sep. 2006.
- [32] <http://www.inc.cuhk.edu.hk/research/projectsphysical-layer-network-coding-pnc>
- [33] C. Fragouli, J. Y. Le Boudec, and J. Widmer, "Network coding: An instant primer," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 1, pp. 63-68, 2006.
- [34] P. A. Chou, and Y. Wu, "Network coding for the internet and wireless networks," *IEEE Signal Processing Magazine*, vol. 24, no. 5, 2007.
- [35] <http://steinwurf.com/tag/kodo/>
- [36] J. Hansen, D. E. Lucani, J. Krigslund, M. Médard, F. H. P. Fitzek, "Network coded software defined networking: enabling 5G transmission and storage networks," *IEEE Comm. Mag.*, vol. 53, no. 9, pp. 100-107
- [37] D. Szabo, A. Csoma, P. Megyesi, A. Gulyas, and F. H. Fitzek, "Network coding as a service," arXiv preprint arXiv:1601.03201, 2016.
- [38] M. Gastpar and M. Vetterli, *Information Processing in Sensor Networks: Second International Workshop, IPSN 2003, Palo Alto, CA, USA, April 22-23, 2003 Proceedings*, 2003, ch. Source-Channel Communication in Sensor Networks, pp. 162-177.
- [39] G. Mergen and L. Tong, "Type based estimation over multiaccess channels," *IEEE Transactions on Signal Processing*, vol. 54, no. 2, pp. 613-626, Feb 2006.
- [40] W. U. Bajwa, J. D. Haupt, A. M. Sayeed, and R. D. Nowak, "Joint source-channel communication for distributed estimation in sensor networks," *IEEE Transactions on Information Theory*, vol. 53, no. 10, pp. 3629-3653, Oct. 2007.
- [41] S. Stańczak, M. Wiczanowski, and H. Boche, "Distributed utility-based power control: Objectives and algorithms," *IEEE Transactions on Signal Processing*, vol. 55, no. 10, pp. 5058-5068, Oct. 2007.
- [42] M. K. Banavar, C. Tepedelenlioglu, and A. Spanias, "Distributed SNR estimation with power constrained signaling over Gaussian multiple-access channels," *IEEE Transactions on Signal Processing*, vol. 60, no. 6, pp. 3289-3294, June 2012.
- [43] B. Nazer and M. Gastpar, "Computation over multiple-access channels," *IEEE Trans. Info. Theory*, vol. 53, no. 10, pp. 3498-3516, Oct. 2007.
- [44] B. Nazer, and M. Gastpar, "Compute-and-forward: Harnessing interference through structured codes," *IEEE Trans. Info. Theory*, vol. 57, no. 10, pp. 6463-6486, Oct. 2011.
- [45] M. Goldenbaum, H. Boche, and S. Stańczak, "Harnessing interference for analog function computation in wireless sensor networks," *IEEE Trans. Signal Processing*, vol. 61, no. 20, pp. 4893-4906, Oct. 2013.
- [46] K. Miller, T. Biermann, H. Woesner, and H. Karl, "Network coding in passive optical networks," *IEEE International Symp. Network Coding*, Toronto, Ontario, Canada, June 2010, pp. 16.
- [47] K. Fouli, M. Maier, and M. Médard, "Network coding in next-generation passive optical networks," *IEEE Communications Magazine*, vol. 49, no. 9, pp. 38-46, Sept. 2011.
- [48] H. J. Caulfield and S. Dolev, *Nat. Phot.* 4, 261 (2010).
- [49] M. Lukosevicius, H. Jaeger, "Reservoir computing approaches to recurrent neural network training," *Computer Science Review*, vol. 3, no. 3, pp. 127-149, Aug. 2009.
- [50] L. Appeltant, M. C. Soriano, G. Van der Sande, J. Danckaert, S. Massar, J. B. Dambre, C. R. Mirasso, and I. Fischer, "Information processing using a single dynamical node as complex system," *Nature Commun.* 2, 468, 2011.
- [51] L. Larger, M.C. Soriano, D. Brunner, L. Appeltant, J.M. Gutierrez, L. Pesquera, C.R. Mirasso, I. Fischer, "Photonic information processing beyond Turing: an optoelectronic implementation of reservoir computing," *Optics Express* 20(3), 3241-3249, 2012.
- [52] Y. Paquot, F. Duport, A. Smerieri, J. Dambre, B. Schrauwen, M. Haelterman, S. Massar, "Optoelectronic reservoir computing," *Scientific Reports* 2, 287, 2012.
- [53] D. Brunner, M. C. Soriano, C. R. Mirasso, and I. Fischer, "Parallel photonic information processing at Gbps data rates using transient states," *Nature Communications* 4, 1364, 2013.
- [54] K. Vandoorne, S. Member, J. Dambre, D. Verstraeten, B. Schrauwen, and P. Bienstman, "Parallel RC using optical amplifiers," *IEEE Trans. Neural Nets*, vol. 22, no. 9, 2011.
- [55] M. Hermans, M.C. Soriano, J. Dambre, P. Bienstman, I. Fischer, "Photonic delay systems as machine learning implementations", *J. Mach. Learn. Res.* 16, 2081, 2015.
- [56] J. W. Lockwood, N. McKeown, G. Watson, G. Gibb, P. Hartke, J. Naoas, R. Raghuraman, and J. Luo, "NetFPGA—an open platform for gigabit-rate network switching and routing," *IEEE International Conference on Microelectronic Systems Education*, pp. 160-161, 2007.
- [57] S. Kim, W. S. Jeong, W. W. Ro, and J. L. Gaudiot, "Design and evaluation of random linear network coding Accelerators on FPGAs," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 13, no. 1, pp. 13, 2013.
- [58] S. M. Choi, K. Lee, and J. Park, "Massive parallelization for random linear network coding," *Appl. Math. Inf. Sci.*, vol. 9, no. 2L, pp. 571-578, 2013.
- [59] S. Kannan, "Layering principles for wireless networks," Ph.D. dissertation, University of Illinois at Urbana-Champaign, 2012.
- [60] R. Kötter and M. Médard, "An algebraic approach to network coding," *IEEE/ACM Trans. Networking*, vol. 11, no. 5, pp. 782-795, Oct. 2003.
- [61] H. Kowshik and P.R. Kumar, "Optimal computation of symmetric boolean functions in collocated networks," *IEEE J. Selected Areas in Communications*, vol. 31, no. 4, pp. 639-654, Apr. 2013.
- [62] V. Lalitha, N. Prakash, K. Vinodh, P. Vijay Kumar, and S. Sandeep Pradhan, "Linear coding schemes for the distributed computation of subspaces," *IEEE J. Selected Areas in Communications*, vol. 31, no. 4, pp. 678-690, Apr. 2013.
- [63] B.K. Rai and B.K. Dey, "On network coding for sum-networks," *IEEE Transactions on Information Theory*, vol. 58, no. 1, pp. 50-63, Jan. 2012.
- [64] A. Ramamoorthy and M. Langberg, "Communicating the sum of sources over a network," *IEEE Journ. Sel. Areas in Comm's*, vol. 31, no. 4, pp. 655-665, Apr. 2013.
- [65] M. Sefidgaran and A. Tchamkerten, "Distributed function computation over a rooted directed tree," to appear in *IEEE Trans. Info. Theory*.
- [66] V. Shah, B.K. Dey, and D. Manjunath, "Network flows for function computation," *IEEE Journ. Sel. Areas in Comm's*, vol. 31, no. 4, pp. 714-730, Apr. 2013.
- [67] F. H. Fitzek, T. Toth, A. Szabados, M. V. Pedersen, D. E. Lucani, M. Sipos, H. Charaf and M. Médard, "Implementation and performance evaluation of distributed cloud storage solutions using random linear network coding," *IEEE ICC Workshops 2014*, pp. 249-254, Sydney, Australia.
- [68] C. Fragouli, and E. Soljanin, "Network coding: Fundamentals and applications," *Foundations and Trends in Networking*, vol. 2, no. 1, 2007.
- [69] T. Ho, M. Médard, R. Koetter, D. R. Karger, M. Effros, J. Shi, and B. Leong, "A random linear network coding approach to multicast," *IEEE Trans. Info. Theory*, vol. 52, no. 10, pp. 4413-4430, Oct. 2006.
- [70] E. Candes and M. Wakin, "An introduction to compressive sampling," *IEEE Signal Processing Magazine*, vol. 25, pp. 21-30, March 2008.
- [71] K. Hayashi, M. Nagahara, and T. Tanaka, "A user's guide to compressed sensing for communications systems," *IEICE Trans. on Communications*, vol. E96-B, no. 3, pp. 685-712, Mar. 2013.
- [72] M. I. Jordan, and T. Mitchell, "Machine learning: Trends, perspectives, and prospects," *Science*, 349, pp. 255-260, 2015.
- [73] Y. Zhang, J. C. Duchi, and M. J. Wainwright, "Communication-efficient algorithms for statistical optimization," *Journal of Machine Learning Research*, vol. 14, pp. 3321-3363, Nov. 2013.
- [74] A. Agarwal, J. Duchi, "Distributed delayed stochastic optimization," *proc. Advances in Neural Information Processing Systems*, 2011.
- [75] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Found. Trends in Machine Learning*, vol. 3, no. 1, 2011.
- [76] O. Shamir, N. Srebro, T. Zhang, "Communication efficient distributed optimization using an approximate Newton-type method," *31st International Conference on Machine Learning, ICML*, 2014.
- [77] L. Wasserman, "All of statistics," Springer, 2004.

- [78] L. Bottou and O. Bousquet, "The tradeoffs of large scale learning," in *Optimization for Machine Learning*, 351-368, MIT Press, 2011.
- [79] F. Yousefian, A. Nedic, U. V. Shanbhag, "On stochastic gradient and subgradient methods with adaptive steplength sequences," *Automatica*, vol. 48, no. 1, pp. 56-67, 2012.
- [80] F. Niu, B. Recht, C. Re, S. J. Wright, "HOGWILD!: A lockFree approach to parallelizing stochastic gradient descent," available at: <http://arxiv.org/abs/1106.5730>.
- [81] L. Bottou, "Large-scale machine learning with stochastic gradient descent," *19th Int'l Conf. on Comp. Statistics (COMPSTAT 2010)*, Paris, France, pp. 177-187, Aug. 2010.
- [82] R. Olfati-Saber, J. A. Fax, and R. M. Murray, "Consensus and cooperation in networked multi-agent systems," *Proc. of the IEEE*, vol. 95, no. 1, pp. 215-233, Jan. 2007.
- [83] M. H. DeGroot, "Reaching a consensus," *Journal of the American Statistical Association*, vol. 69, no. 345, March 1974.
- [84] S. Kar and J. M. F. Moura, "Distributed consensus algorithms in sensor networks: link failures and channel noise," *IEEE Trans. Signal Processing*, vol. 57, no. 1, pp. 355-369, Jan. 2009.
- [85] E. Fasolo, M. Rossi, J. Widmer, and M. Zorzi, "In-network aggregation techniques for wireless sensor networks: a survey," *IEEE Trans. Wireless Comm's*, vol. 14, no. 2, pp. 70-87, Apr. 2007.
- [86] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. M. Hellerstein, "Graphlab: A new framework for parallel machine learning," *Uncertainty in Artificial Intelligence*, 2010.
- [87] M. Bhandarkar, "MapReduce programming with apache Hadoop," *IEEE Parallel and Dist. Processing (IPDPS)*, 2010.
- [88] <http://www.etsi.org/technologies-clusters/technologies/mobile-edge-computing>